

INTAREA
Internet-Draft
Intended status: Standards Track
Expires: April 9, 2017

R. Bonica
R. Thomas
Juniper Networks
October 6, 2016

Extended Ping (Xping)
draft-bonica-intarea-eping-02

Abstract

This document describes a new diagnostic tool called Extended Ping (Xping). Network operators execute Xping to determine the status of a remote interface. In this respect, Xping is similar to Ping. Xping differs from Ping in that it does not require network reachability between itself and remote interface whose status is being queried.

Xping relies on two new ICMP messages, called Extended Echo and Extended Echo Reply. Both ICMP messages are defined herein.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 9, 2017.

Copyright Notice

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

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

Table of Contents

1. Problem Statement	2
2. ICMP Extended Echo	4
2.1. Interface Identification Object	6
3. ICMP Extended Echo Reply	7
4. ICMP Extended Echo and Extended Echo Reply Processing	8
4.1. Code Field Processing	9
5. The Eping Application	10
6. IANA Considerations	11
7. Security Considerations	12
8. Acknowledgements	12
9. References	12
9.1. Normative References	12
9.2. Informative References	13
Appendix A. An Appendix	14
Authors' Addresses	14

1. Problem Statement

Network operators use Ping [RFC2151] to determine whether a remote interface is alive. Ping sends an ICMP [RFC0792] [RFC4443] Echo message to the interface being probed and waits for an ICMP Echo Reply. If Ping receives the expected ICMP Echo Reply, it reports that the probed interface is alive.

In order for the ICMP Echo message to reach the probed interface, the probed interface must be addressed appropriately. IP addresses are scoped as follows:

- o Global [RFC4291]
- o Private [RFC1918] [RFC4193]

- o Link-local [RFC3927] [RFC4291]

Global addresses are the most widely scoped. A globally addressed interface can be reached from any node on the Internet. By contrast, link-local addresses are the least widely scoped. An interface whose only address is link-local can be reached from on-link interfaces only.

Network operators seek to decrease their dependence on widely-scoped interface addressing. For example:

- o The operator of an IPv4 network currently assigns global addresses to all interfaces. In order to conserve scarce IPv4 address space, this operator seeks to renumber selected interfaces with private addresses.
- o The operator of an IPv4 network currently assigns private addresses to all interfaces. In order to achieve operational efficiencies, this operator seeks to leave selected interfaces unnumbered.
- o The operator of an IPv6 network currently assigns global addresses to all interfaces. In order to achieve operational efficiencies, this operator seeks to number selected interfaces with link-local addresses only.

When a network operator rennumbers an interface, replacing a more widely scoped address with one that is less widely scoped, the operator also reduces the number of nodes from which Ping can probe the interface. Therefore, many network operators who rely on Ping remain dependant upon widely scoped interface addressing.

This document describes a new diagnostic tool called Extended Ping (Xping). Network operators use Xping to determine the status of a remote interface. In this respect, Xping is similar to Ping. Xping differs from Ping in that it does not require reachability between the probing node and the probed interface. Or, said another way, Xping does not require reachability between the node upon which it executes and the interface whose status is being queried.

Xping relies on two new ICMP messages, called Extended Echo and Extended Echo Reply. The Extended Echo message makes a semantic distinction between the destination interface and the probed interface. The destination interface is the interface to which the Extended Echo message is delivered. It must be reachable from the probing node. The probed interface is the interface whose status is being queried. It does not need to be reachable from the probing

node. However, the destination and probed interfaces must be local to one another (i.e., the same node must support both interfaces).

Because the Extended Echo message makes a distinction between the destination and probed interfaces, Xping can probe every interface on a node if it can reach any interface on the node. In many cases, this allows network operators to decrease their dependence on widely scoped interface addressing.

Network operators can use Xping to determine the operational status of the probed interface. They can also use Xping to determine which protocols (e.g., IPv4, IPv6) are active on the interface. However, they cannot use Xping to obtain other information regarding the interface (e.g., bandwidth, MTU). In order to obtain such information, they should use other network management protocols (e.g., SNMP, Netconf).

This document is divided into sections, with Section 2 describing the Extended Echo message and Section 3 describing the Extended Echo Reply message. Section 4 describes how the probed node processes the Extended Echo message and Section 5 describes the Xping application.

2. ICMP Extended Echo

The ICMP Extended Echo message is applicable to both ICMPv4 and ICMPv6. Like any ICMP message, the ICMP Extended Echo message is encapsulated in an IP header. The ICMPv4 version of the Extended Echo message is encapsulated in an IPv4 header, while the ICMPv6 version is encapsulated in an IPv6 header.

Figure 1 depicts the ICMP Extended Echo message.

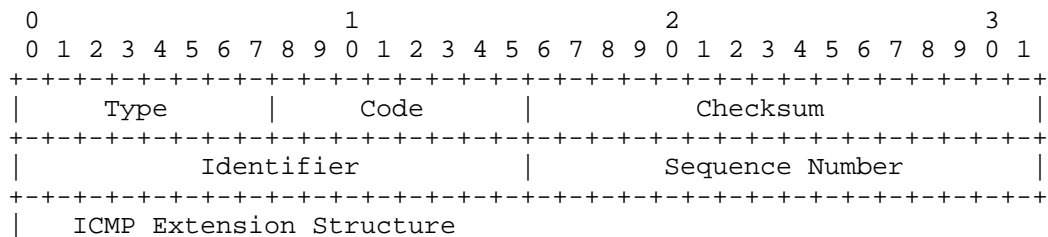


Figure 1: ICMP Extended Echo Message

IP Header fields:

- o Source Address: Identifies an interface on the probing node.
- o Destination Address: Identifies the destination interface (i.e., the interface to which this message will be delivered).

ICMP fields:

- o Type: Extended Echo. The value for ICMPv4 is TBD by IANA. The value for ICMPv6 is also TBD by IANA.
- o Code: 0
- o Checksum: For ICMPv4, see RFC 792. For ICMPv6, see RFC 4443.
- o Identifier: An identifier to aid in matching Extended Echo Replies to this Extended Echo Request. May be zero.
- o Sequence Number: A sequence number to aid in matching Extended Echo Replies to this Extended Echo Request. May be zero.
- o ICMP Extension Structure: Identifies the probed interface, by name, index or address.

If the ICMP Extension Structure identifies the probed interface by address, that address can be a member of any address family. For example:

- o An ICMPv4 Extended Echo message can carry an ICMP Extension Structure that identifies the probed interface by IPv4 address
- o An ICMPv4 Extended Echo message can carry an ICMP Extension Structure that identifies the probed interface by IPv6 address
- o An ICMPv6 Extended Echo message can carry an ICMP Extension Structure that identifies the probed interface by IPv4 address
- o An ICMPv6 Extended Echo message can carry an ICMP Extension Structure that identifies the probed interface by IPv6 address

Section 7 of [RFC4884] defines the ICMP Extension Structure. As per RFC 4884, the Extension Structure contains exactly one Extension Header followed by one or more objects. When applied to the ICMP Extended Echo message, the ICMP Extension Structure contains one or two instances of the Interface Identification Object (Section 2.1).

In most cases, a single instance of the Interface Identification Object can identify the probed interface. However, two instance are

required when neither uniquely identifies a interface (e.g., an IPv6 link-local address and an IEEE 802 address).

2.1. Interface Identification Object

The Interface Identification Object identifies the probed interface by name, index, or address. Like any other ICMP Extension Object, it contains an Object Header and Object Payload. The Object Header contains the following fields:

- o Class-Num: Interface Identification Object. Value is TBD by IANA
- o C-type: Values are: (1) Identifies Interface By Name, (2) Identifies Interface By Index, and (3) Identifies Interface By Address
- o Length: Length of the object, measured in octets, including the object header and object payload.

If the Interface Identification Object identifies the probed interface by name, the object payload contains the human-readable interface name. The interface name SHOULD be the full MIB-II ifName [RFC2863], if less than 255 octets, or the first 255 octets of the ifName, if the ifName is longer. The interface name MAY be some other human-meaningful name of the interface. The interface name MUST be represented in the UTF-8 charset [RFC3629] using the Default Language [RFC2277].

If the Interface Identification Object identifies the probed interface by index, the length is equal to 8 and the payload contains the MIB-II ifIndex [RFC 2863].

If the Interface Identification Object identifies the probed interface by address, the payload is as depicted in Figure 2.

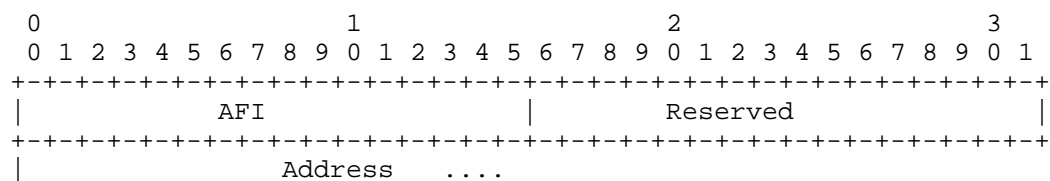


Figure 2: Interface Identification Object - C-type 3 Payload

Payload fields are defined as follows:

- o Address Family Identifier (AFI): This 16-bit field identifies the type of address represented by the Address field. All values found in the IANA registry of Address Family Numbers (available from <<http://www.iana.org>>) are valid in this field. Implementations MUST support values (1) IPv4, (2) IPv6 and (6) IEEE 802. They MAY support other values.
- o Reserved: This 16-bit field MUST be set to zero and ignored upon receipt.
- o Address: This variable-length field represents an address associated with the probed interface.

3. ICMP Extended Echo Reply

The ICMP Extended Echo Reply message is applicable to both ICMPv4 and ICMPv6. Like any ICMP message, the ICMP Extended Echo Reply message is encapsulated in an IP header. The ICMPv4 version of the Extended Echo Reply message is encapsulated in an IPv4 header, while the ICMPv6 version is encapsulated in an IPv6 header.

Figure 3 depicts the ICMP Extended Echo Reply message.

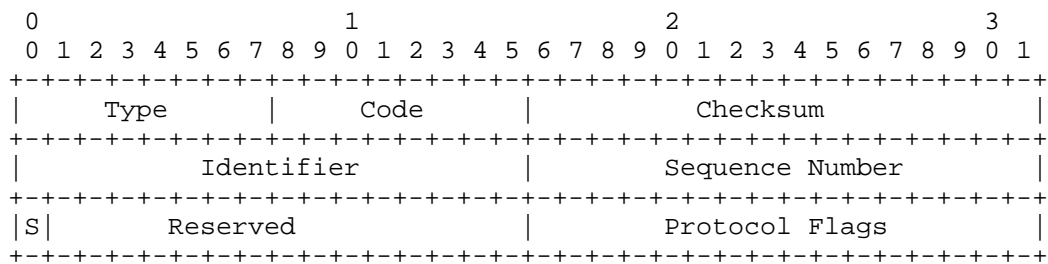


Figure 3: ICMP Extended Echo Reply Message

IP Header fields:

- o Source address: Identifies the interface to which the corresponding ICMP Extended Echo message was sent
- o Destination address: Identifies the interface from which the corresponding ICMP Extended Echo message was sent

ICMP fields:

- o Type: Extended Echo Reply. The value for ICMPv4 is TBD by IANA. The value for ICMPv6 is also TBD by IANA.
- o Code: (0) No Error, (1) Xping Not Enabled, (2) Malformed Query, (3) Query Type Not Enabled, (4) No Such Interface, (5) Multiple Interfaces Satisfy Query
- o Checksum: For ICMPv4, see RFC 792. For ICMPv6, see RFC 4443.
- o Identifier: An identifier to aid in matching Extended Echo Replies to this Extended Echo Request. May be zero.
- o Sequence Number: A sequence number to aid in matching Extended Echo Replies to this Extended Echo Request. May be zero.
- o S Bit: This bit is set if the Code field is equal to No Error (0) and the probed interface is active. Otherwise, this bit is clear.
- o Reserved: This 15-bit field MUST be set to zero and ignored upon receipt.
- o Protocol Flags: Each bit in this field represents a protocol. The bit is set if the S-bit is set and the corresponding protocol is running on the probed interface. Bit mappings are as follows: Bit 0 (IPv4), Bit 1 (IPv6), Bit 2 (Ethernet), Bits 3-15 (Reserved)

4. ICMP Extended Echo and Extended Echo Reply Processing

When a node receives an ICMPv4 Extended Echo, it MUST format an ICMP Extended Echo Reply as follows:

- o Don't Fragment flag (DF) is 1
- o More Fragments flag is 0
- o Fragment Offset is 0
- o TTL is 255
- o Protocol is ICMP

When a node receives an ICMPv6 Extended Echo, it MUST format an ICMPv6 Extended Echo Reply as follows:

- o Hop Limit is 255
- o Next Header is ICMPv6

In either case, the responding node MUST:

- o Copy the source address from the Extended Echo message to the destination address of the Extended Echo Reply
- o Copy the destination address from the Extended Echo message to the source address of the Extended Echo Reply
- o Set the DiffServ codepoint to CS0 [RFC4594]
- o Set the ICMP Type to Extended Echo Reply
- o Copy the Identifier from the Extended Echo message to the Extended Echo Reply
- o Copy the sequence number from the Extended Echo message to the Extended Echo Reply
- o Set the Code field as described Section 4.1
- o If the Code Field is equal to No Error (0) and the probed interface is active, set the S-Bit. Otherwise, clear the S-Bit.
- o If the S-bit is set, set Protocol Flags as appropriate. Otherwise, clear all Protocol Flags.
- o Set the checksum appropriately
- o Forward the ICMP Extended Echo Reply to its destination

4.1. Code Field Processing

The following rules govern how the Code should be set:

- o If Xping is not enabled, set the Code to Xping Not Enabled (1)
- o Otherwise, if the query is malformed, set the Code to Malformed Query (2)
- o Otherwise, if the query type is not supported, set the Code to Query Type Not Enabled (3)
- o Otherwise, if the ICMP Extension Structure does not identify any local interfaces, set the Code to No Such Interface (4)
- o Otherwise, if the ICMP Extension Structure identifies more than one local interfaces, set the Code to Multiple Interfaces Satisfy Query (5)

- o Otherwise, set the code to No Error (0)

5. The Eping Application

The Xping application accepts input parameters, sets a counter and enters a loop to be exited when the counter is equal to zero. On each iteration of the loop, Xping emits an ICMP Extended Echo, decrements the counter, sets a timer, waits for the timer to expire. If an expected ICMP Extended Echo Reply arrives while Xping is waiting for the timer to expire, Xping relays information returned by that message to its user. However, on each iteration of the loop, Xping waits for the timer to expire, regardless of whether an Extended Echo Reply message arrives.

Xping accepts the following parameters:

- o Count
- o Wait
- o Source Interface Address
- o Hop Count
- o Destination Interface Address
- o Probed Interface Identifier

Count is a positive integer whose default value is 3. Count determines the number of times that Xping iterates through the above-mentioned loop.

Wait is a positive integer whose minimum and default values are 1. Wait determines the duration of the above-mentioned timer, measured in seconds.

Source Interface Address specifies the source address of ICMP Extended Echo. The source address MUST identify an interface that is local to the probing node.

The destination Interface Address identifies the interface to which the ICMP Extended Echo message is sent. It can be an IPv4 address or an IPv6 address. If it is an IPv4 address, Xping emits an ICMPv4 message. If it is an IPv6 address, Xping emits an ICMPv6 message.

The probed interface is the interface whose status is being queried. If the probed interface identifier is not specified, the Xping application invokes the traditional Ping application and terminates.

If the probed interface identifier is specified, it can be any of the following:

- o an interface name
- o an address from any address family (e.g., IPv4, IPv6, MAC)
- o an ifIndex

The probed interface identifier can have any scope. For example, the probed interface identifier can be:

- o an IPv6 address, whose scope is global
- o an IPv6 address, whose scope is link-local
- o an interface name, whose scope is node-local
- o an ifIndex, whose scope is node-local

If the probed interface identifier is an address, it does not need to be of the same address family as the destination interface address. For example, Xping accepts an IPv4 destination interface address and an IPv6 probed interface identifier.

6. IANA Considerations

This document requests the following actions from IANA:

- o Add an entry to the "ICMP Type Number" registry, representing the Extended Echo. This entry has one code (0).
- o Add an entry to the "Internet Control Message Protocol version 6 (ICMPv6) Parameters" registry, representing the Extended Echo. This entry has one code (0).
- o Add an entry to the "ICMP Type Number" registry, representing the Extended Echo Reply. This entry has the following codes: (0) No Error, (1) Xping Not Enabled, (2) Malformed Query, (3) Query Type Not Enabled, (4) No Such Interface, (5) Multiple Interfaces Satisfy Query. Protocol Flag Bit mappings are as follows: Bit 0 (IPv4), Bit 1 (IPv6), Bit 2 (Ethernet), Bits 3-15 (Reserved).
- o Add an entry to the "Internet Control Message Protocol version 6 (ICMPv6) Parameters" registry, representing the Extended Echo Reply. This entry has the following codes: (0) No Error, (1) Xping Not Enabled, (2) Malformed Query, (3) Query Type Not Enabled, (4) No Such Interface, (5) Multiple Interfaces Satisfy

Query. Protocol Flag Bit mappings are as follows: Bit 0 (IPv4), Bit 1 (IPv6), Bit 2 (Ethernet), Bits 3-15 (Reserved).

- o Add an entry to the "ICMP Extension Object Classes and Class Subtypes" registry, representing the Interface Identification Object. It has C-types Reserved (0), Identifies Interface By Name (1), Identifies Interface By Index (2), Identifies Interface By Address (3)

Note to RFC Editor: this section may be removed on publication as an RFC.

7. Security Considerations

Implementations MUST include a configuration option that enables processing of the ICMP Extended Echo. By default, this configuration option MUST be disabled. When an implementation receives an ICMP Extended Echo and this configuration option is disabled, the implementation returns an ICMP Extended Reply with Code equal to Xping Not Enabled (1).

Implementations MUST also include a configuration options that enable the probed interface identification by name, index and address. By default, these configuration options MUST be enabled. When an implementation receives an ICMP Extended Echo and the appropriate configuration option is disabled, the implementation returns an ICMP Extended Reply with Code equal to Query Type Not Enabled (3).

8. Acknowledgements

Thanks to Jeff Haas, Carlos Pignataro and Joe Touch for their thoughtful review of this document.

9. References

9.1. Normative References

- [RFC0792] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, DOI 10.17487/RFC0792, September 1981, <<http://www.rfc-editor.org/info/rfc792>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.

- [RFC2277] Alvestrand, H., "IETF Policy on Character Sets and Languages", BCP 18, RFC 2277, DOI 10.17487/RFC2277, January 1998, <<http://www.rfc-editor.org/info/rfc2277>>.
- [RFC2863] McCloghrie, K. and F. Kastenholz, "The Interfaces Group MIB", RFC 2863, DOI 10.17487/RFC2863, June 2000, <<http://www.rfc-editor.org/info/rfc2863>>.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <<http://www.rfc-editor.org/info/rfc3629>>.
- [RFC4443] Conta, A., Deering, S., and M. Gupta, Ed., "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", RFC 4443, DOI 10.17487/RFC4443, March 2006, <<http://www.rfc-editor.org/info/rfc4443>>.
- [RFC4884] Bonica, R., Gan, D., Tappan, D., and C. Pignataro, "Extended ICMP to Support Multi-Part Messages", RFC 4884, DOI 10.17487/RFC4884, April 2007, <<http://www.rfc-editor.org/info/rfc4884>>.

9.2. Informative References

- [RFC1918] Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, DOI 10.17487/RFC1918, February 1996, <<http://www.rfc-editor.org/info/rfc1918>>.
- [RFC2151] Kessler, G. and S. Shepard, "A Primer On Internet and TCP/IP Tools and Utilities", FYI 30, RFC 2151, DOI 10.17487/RFC2151, June 1997, <<http://www.rfc-editor.org/info/rfc2151>>.
- [RFC3927] Cheshire, S., Aboba, B., and E. Guttman, "Dynamic Configuration of IPv4 Link-Local Addresses", RFC 3927, DOI 10.17487/RFC3927, May 2005, <<http://www.rfc-editor.org/info/rfc3927>>.
- [RFC4193] Hinden, R. and B. Haberman, "Unique Local IPv6 Unicast Addresses", RFC 4193, DOI 10.17487/RFC4193, October 2005, <<http://www.rfc-editor.org/info/rfc4193>>.
- [RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <<http://www.rfc-editor.org/info/rfc4291>>.

[RFC4594] Babiarz, J., Chan, K., and F. Baker, "Configuration Guidelines for DiffServ Service Classes", RFC 4594, DOI 10.17487/RFC4594, August 2006, <<http://www.rfc-editor.org/info/rfc4594>>.

Appendix A. An Appendix

Authors' Addresses

Ron Bonica
Juniper Networks
2251 Corporate Park Drive
Herndon, Virginia 20171
USA

Email: rbonica@juniper.net

Reji Thomas
Juniper Networks
Elnath-Exora Business Park Survey
Bangalore, Kanata 560103
India

Email: rejithomas@juniper.net

INTERNET-DRAFT
Intended Status: Proposed Standard
Expires: May 1, 2017

T. Herbert
Facebook
L. Yong
Huawei
F. Templin
Boeing

October 28, 2016

Extensions for Generic UDP Encapsulation
draft-herbert-gue-extensions-01

Abstract

This specification defines a set of the fundamental optional extensions for Generic UDP Encapsulation (GUE). The extensions defined in this specification are the security option, payload transform option, checksum option, fragmentation option, and the remote checksum offload option.

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/lid-abstracts.html>

The list of Internet-Draft Shadow Directories can be accessed at
<http://www.ietf.org/shadow.html>

Copyright and License Notice

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

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

Table of Contents

1. Introduction	4
2. GUE header format with optional extensions	4
3. Security option	5
3.1. Extension field format	6
3.2. Usage	6
3.3. Cookies	7
3.4. HMAC	7
3.4.1. Extension field format	7
3.4.2. Selecting a hash algorithm	8
3.4.3. Pre-shared key management	8
3.5. Interaction with other optional extensions	9
4. Fragmentation option	9
4.1. Motivation	9
4.2. Scope	11
4.3. Extension field format	11
4.4. Fragmentation procedure	12
4.5. Reassembly procedure	14
4.6. Security Considerations	16
5. Payload transform option	16
5.1. Extension field format	16
5.2. Usage	17
5.3. Interaction with other optional extensions	17
5.4. DTLS transform	18
6. Remote checksum offload option	18
6.1. Extension field format	19
6.2. Usage	19
6.2.1. Transmitter operation	19
6.2.2. Receiver operation	20
6.3. Security Considerations	21
7. Checksum option	21
7.1. Extension field format	21
7.2. Requirements	22
7.3. GUE checksum pseudo header	22
7.4. Usage	24
7.4.1. Transmitter operation	24

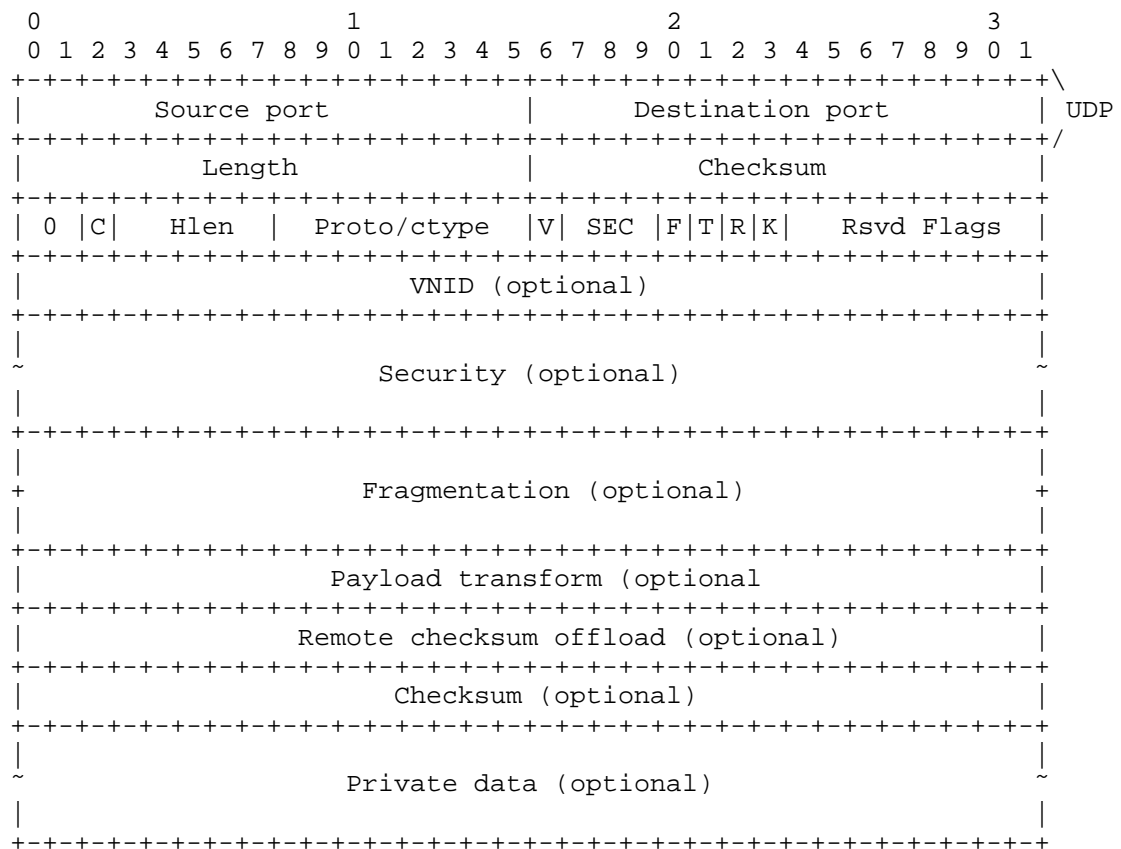
7.4.2.Receiver operation	24
7.5. Security Considerations	25
8. Processing order of options	25
9. Security Considerations	26
10. IANA Consideration	27
11. References	27
11.1. Normative References	27
11.2. Informative References	28
Authors' Addresses	29

1. Introduction

Generic UDP Encapsulation (GUE) [I.D.nvo3-gue] is a generic and extensible encapsulation protocol. This specification defines a fundamental set of optional extensions for version 0 of GUE. These extensions are the security option, payload transform option, checksum option, fragmentation option, and the remote checksum offload option.

2. GUE header format with optional extensions

The format of a version 0 GUE header with the optional extensions defined in this specification is:



The contents of the UDP header are described in [I.D.herbert-gue].

The GUE header consists of:

- o Ver: Version. Set to 0 to indicate GUE encapsulation header. Note that version 1 does not allow options.
- o C: C-bit. Indicates the GUE payload is a control message when set, a data message when not set. GUE optional extensions can be used with either control or data messages unless otherwise specified in the option definition.
- o Hlen: Length in 32-bit words of the GUE header, including optional extension fields but not the first four bytes of the header. Computed as $(\text{header_len} - 4) / 4$. The length of the encapsulated packet is determined from the UDP length and the Hlen: $\text{encapsulated_packet_length} = \text{UDP_Length} - 12 - 4 * \text{Hlen}$.
- o Proto/ctype: If the C-bit is not set this indicates IP protocol number for the packet in the payload; if the C bit is set this is the type of control message in the payload. The next header begins at the offset provided by Hlen. When the payload transform option or fragmentation option is used this field may be set to protocol number 59 for a data message, or zero for a control message, to indicate no next header for the payload.
- o V: Indicates the network virtualization extension (VNID) field is present. The VNID option is described in [I.D.hy-nvo3-gue-4-nvo].
- o SEC: Indicates security extension field is present. The security option is described in section 3.
- o F: Indicates fragmentation extension field is present. The fragmentation option is described in section 4.
- o T: Indicates payload transform extension field is present. The payload transform option is described in section 5.
- o R: Indicates the remote checksum extension field is present. The remote checksum offload option is described in section 6.
- o K: Indicates checksum extension field is present. The checksum option is described in section 7.
- o Private data is described in [I.D.nvo3-gue].

3. Security option

The GUE security option provides origin authentication and integrity

3.1. Extension field format

The format of the security option is:

The fields of the option are:

- To provide security capability, the SEC flags MUST be set. Different sizes are allowed to allow different methods and extensibility. The use of the security field is expected to be negotiated out of band between two tunnel end points.

The values in the SEC flags are:

- o 000b - No security field
- o 001b - 64 bit security field
- o 010b - 128 bit security field
- o 011b - 256 bit security field
- o 100b - 388 bit security field (HMAC)
- o 101b, 110b, 111b - Reserved values

3.2. Usage

[Page 6]

negotiated out of band between two communicating hosts. Two security algorithms are defined below.

3.3. Cookies

The security field may be used as a cookie. This would be similar to the cookie mechanism described in L2TP [RFC3931], and the general properties should be the same. A cookie may be used to validate the encapsulation. The cookie is a shared value between an encapsulator and decapsulator which should be chosen randomly and may be changed periodically. Different cookies may be used for logical flows between the encapsulator and decapsulator, for instance packets sent with different VNIDs in network virtualization [I.D.hy-nvo3-gue-4-nvo] might have different cookies. Cookies may be 64, 128, or 256 bits in size.

3.4. HMAC

Key-hashed message authentication code (HMAC) is a strong method of checking integrity and authentication of data. This sections defines a GUE security option for HMAC. Note that this is based on the HMAC TLV description in "IPv6 Segment Routing Header (SRH)" [I.D.previdi-6man-sr-header].

3.4.1. Extension field format

The HMAC option is a 288 bit field (36 octets). The security flags are set to 100b to indicates the presence of a 288 bit security field.

The format of the field 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
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     HMAC Key-id                                     |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     HMAC (256 bits)                                     |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Fields are:

- o HMAC Key-id: opaque field to allow multiple hash algorithms or key selection
- o HMAC: Output of HMAC computation

The HMAC field is the output of the HMAC computation (per RFC 2104 [RFC2104]) using a pre-shared key identified by HMAC Key-id and of the text which consists of the concatenation of:

- o The IP addresses
- o The GUE header including all private data and all optional extensions that are present except for the security option

The purpose of the HMAC option is to verify the validity, the integrity and the authorization of the GUE header itself.

The HMAC Key-id field allows for the simultaneous existence of several hash algorithms (SHA-256, SHA3-256 ... or future ones) as well as pre-shared keys. The HMAC Key-id field is opaque, i.e., it has neither syntax nor semantic. Having an HMAC Key-id field allows for pre-shared key roll-over when two pre-shared keys are supported for a while GUE endpoints converge to a fresher pre-shared key.

3.4.2. Selecting a hash algorithm

The HMAC field in the HMAC option is 256 bit wide. Therefore, the HMAC MUST be based on a hash function whose output is at least 256 bits. If the output of the hash function is 256, then this output is simply inserted in the HMAC field. If the output of the hash function is larger than 256 bits, then the output value is truncated to 256 by taking the least-significant 256 bits and inserting them in the HMAC field.

GUE implementations can support multiple hash functions but MUST implement SHA-2 [FIPS180-4] in its SHA-256 variant.

3.4.3. Pre-shared key management

The field HMAC Key-id allows for:

- o Key roll-over: when there is a need to change the key (the hash pre-shared secret), then multiple pre-shared keys can be used simultaneously. A decapsulator can have a table of <HMAC Key-id, pre-shared secret> for the currently active and future keys.
- o Different algorithms: by extending the previous table to <HMAC Key-id, hash function, pre-shared secret>, the decapsulator can also support simultaneously several hash algorithms (see section Section 5.2.1)

The pre-shared secret distribution can be done:

- o In the configuration of the endpoints
- o Dynamically using a trusted key distribution such as [RFC6407]

The intent of this document is NOT to define yet-another-key-distribution-protocol.

3.5. Interaction with other optional extensions

If GUE fragmentation (section 4) is used in concert with the GUE security option, the security option processing is performed after fragmentation at the encapsulator and before reassembly at the decapsulator.

The GUE payload transform option (section 5) may be used in concert with the GUE security option. The payload transform option could be used to encrypt the GUE payload to provide privacy for an encapsulated packet during transit. The security option provides authentication and integrity for the GUE header (including the payload transform field in the header). The two functions are processed separately at tunnel end points. A GUE tunnel can use both functions or use one of them. Section 5.3 details handling for when both are used in a packet.

4. Fragmentation option

The fragmentation option allows an encapsulator to perform fragmentation of packets being ingress to a tunnel. Procedures for fragmentation and reassembly are defined in this section. This specification adapts the procedures for IP fragmentation and reassembly described in [RFC0791] and [RFC2460]. Fragmentation may be performed on both data and control messages in GUE.

4.1. Motivation

This section describes the motivation for having a fragmentation option in GUE.

MTU and fragmentation issues with In-the-Network Tunneling are described in [RFC4459]. Considerations need to be made when a packet is received at a tunnel ingress point which may be too large to traverse the path between tunnel endpoints.

There are four suggested alternatives in [RFC4459] to deal with this:

- 1) Fragmentation and Reassembly by the Tunnel Endpoints
- 2) Signaling the Lower MTU to the Sources

3) Encapsulate Only When There is Free MTU

4) Fragmentation of the Inner Packet

Many tunneling protocol implementations have assumed that fragmentation should be avoided, and in particular alternative #3 seems preferred for deployment. In this case, it is assumed that an operator can configure the MTUs of links in the paths of tunnels to ensure that they are large enough to accommodate any packets and required encapsulation overhead. This method, however, may not be feasible in certain deployments and may be prone to misconfiguration in others.

Similarly, the other alternatives have drawbacks that are described in [RFC4459]. Alternative #2 implies use of something like Path MTU Discovery which is not known to be sufficiently reliable. Alternative #4 is not permissible with IPv6 or when the DF bit is set for IPv4, and it also introduces other known issues with IP fragmentation.

For alternative #1, fragmentation and reassembly at the tunnel endpoints, there are two possibilities: encapsulate the large packet and then perform IP fragmentation, or segment the packet and then encapsulate each segment (a non-IP fragmentation approach).

Performing IP fragmentation on an encapsulated packet has the same issues as that of normal IP fragmentation. Most significant of these is that the Identification field is only sixteen bits in IPv4 which introduces problems with wraparound as described in [RFC4963].

The second possibility follows the suggestion expressed in [RFC2764] and the fragmentation feature described in the AERO protocol [I.D.templin-aerolink], that is for the tunneling protocol itself to incorporate a segmentation and reassembly capability that operates at the tunnel level. In this method fragmentation is part of the encapsulation and an encapsulation header contains the information for reassembly. This differs from IP fragmentation in that the IP headers of the original packet are not replicated for each fragment.

Incorporating fragmentation into the encapsulation protocol has some advantages:

- o At least a 32 bit identifier can be defined to avoid issues of the 16 bit Identification in IPv4.
- o Encapsulation mechanisms for security and identification, such as virtual network identifiers, can be applied to each segment.
- o This allows the possibility of using alternate fragmentation and

reassembly algorithms (e.g. fragmentation with Forward Error Correction).

- o Fragmentation is transparent to the underlying network so it is unlikely that fragmented packet will be unconditionally dropped as might happen with IP fragmentation.

4.2. Scope

This specification describes the mechanics of fragmentation in Generic UDP Encapsulation. The operational aspects and details for higher layer implementation must be considered for deployment, but are considered out of scope for this document. The AERO protocol [I.D.templin-aerolink] defines one use case of fragmentation with encapsulation.

4.3. Extension field format

The presence of the GUE fragmentation option is indicated by the F bit in the GUE header.

The format of the fragmentation option 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																			
Fragment offset										Res	M	Orig-proto										
Identification																						

The fields of the option are:

- o Fragment offset: This field indicates where in the datagram this fragment belongs. The fragment offset is measured in units of 8 octets (64 bits). The first fragment has offset zero.
- o Res: Two bit reserved field. Must be set to zero for transmission. If set to non-zero in a received packet then the packet MUST be dropped.
- o M: More fragments bit. Set to 1 when there are more fragments following in the datagram, set to 0 for the last fragment.
- o Orig-proto: The control type (when C-bit is set) or the IP protocol (when C-bit is not set) of the fragmented packet.
- o Identification: 40 bits. Identifies fragments of a fragmented

packet.

Pertinent GUE header fields to fragmentation are:

- o C-bit: This is set for each fragment based on the whether the original packet being fragmented is a control or data message.
- o Proto/ctype - For the first fragment (fragment offset is zero) this is set to that of the original packet being fragmented (either will be a control type or IP protocol). For other fragments, this is set to zero for a control message being fragmented, or to "No next header" (protocol number 59) for a data message being fragmented.
- o F bit - Set to indicate presence of the fragmentation extension field.

4.4. Fragmentation procedure

If an encapsulator determines that a packet must be fragmented (eg. the packet's size exceeds the Path MTU of the tunnel) it should divide the packet into fragments and send each fragment as a separate GUE packet, to be reassembled at the decapsulator (tunnel egress).

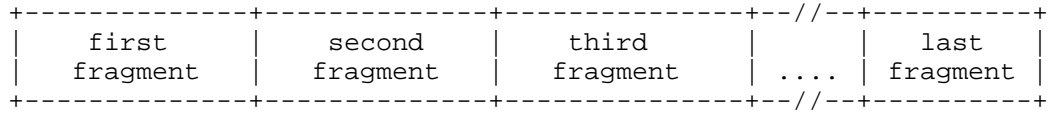
For every packet that is to be fragmented, the source node generates an Identification value. The Identification must be different than that of any other fragmented packet sent within the past 60 seconds (Maximum Segment Lifetime) with the same tunnel identification-- that is the same outer source and destination addresses, same UDP ports, same orig-proto, and same virtual network identifier if present.

The initial, unfragmented, and unencapsulated packet is referred to as the "original packet". This will be a layer 2 packet, layer 3 packet, or the payload of a GUE control message:

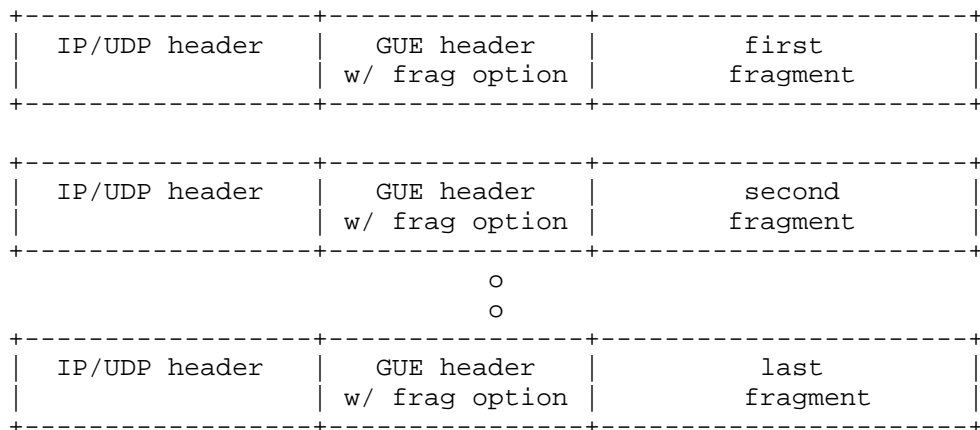
```
+-----//-----+
|               Original packet               |
|      (e.g. an IPv4, IPv6, Ethernet packet)      |
+-----//-----+
```

Fragmentation and encapsulation are performed on the original packet in sequence. First the packet is divided up in to fragments, and then each fragment is encapsulated. Each fragment, except possibly the last ("rightmost") one, is an integer multiple of 8 octets long. Fragments MUST be non-overlapping. The number of fragments should be minimized, and all but the last fragment should be approximately equal in length.

The fragments are transmitted in separate "fragment packets" as:



Each fragment is encapsulated as the payload of a GUE packet. This is illustrated as:



Each fragment packet is composed of:

- (1) Outer IP and UDP headers as defined for GUE encapsulation.
 - o The IP addresses and UDP ports must be the same for all fragments of a fragmented packet.
- (2) A GUE header that contains:
 - o The C-bit which is set to the same value for all the fragments of a fragmented packet based on whether a control message or data message was fragmented.
 - o A proto/ctype. In the first fragment this is set to the value corresponding to the next header of the original packet and will be either an IP protocol or a control type. For subsequent fragments, this field is set to 0 for a fragmented control message or 59 (no next header) for a fragmented data message.
 - o The F bit is set and fragment extension field is present.

- o Other GUE options. Note that options apply to the individual GUE packet. For instance, the security option would be validated before reassembly.
- (3) The GUE fragmentation option. The contents of the extension field include:
- o Orig-proto specifies the protocol of the original packet.
 - o A Fragment Offset containing the offset of the fragment, in 8-octet units, relative to the start of the of the original packet. The Fragment Offset of the first ("leftmost") fragment is 0.
 - o An M flag value of 0 if the fragment is the last ("rightmost") one, else an M flag value of 1.
 - o The Identification value generated for the original packet.
- (4) The fragment itself.

4.5. Reassembly procedure

At the destination, fragment packets are decapsulated and reassembled into their original, unfragmented form, as illustrated:

```
+-----//-----+
|                               |
|               Original packet |
|               (e.g. an IPv4, IPv6, Ethernet packet) |
|                               |
+-----//-----+
```

The following rules govern reassembly:

The IP/UDP/GUE headers of each packet are retained until all fragments have arrived. The reassembled packet is then composed of the decapsulated payloads in the GUE packets, and the IP/UDP/GUE headers are discarded.

When a GUE packet is received with the fragment extension, the proto/ctype field in the GUE header must be validated. In the case that the packet is a first fragment (fragment offset is zero), the proto/ctype in the GUE header must equal the orig-proto value in the fragmentation option. For subsequent fragments (fragment offset is non-zero) the proto/ctype in the GUE header must be 0 for a control message or 59 (no-next-hdr) for a data message. If the proto/ctype value is invalid for a received packet it MUST be dropped.

An original packet is reassembled only from GUE fragment packets that have the same outer source address, destination address, UDP source port, UDP destination port, GUE header C-bit, virtual network identifier if present, orig-proto value in the fragmentation option, and Fragment Identification. The protocol type or control message type (depending on the C-bit) for the reassembled packet is the value of the GUE header proto/ctype field in the first fragment.

The following error conditions may arise when reassembling fragmented packets with GUE encapsulation:

If insufficient fragments are received to complete reassembly of a packet within 60 seconds (or a configurable period) of the reception of the first-arriving fragment of that packet, reassembly of that packet must be abandoned and all the fragments that have been received for that packet must be discarded.

If the payload length of a fragment is not a multiple of 8 octets and the M flag of that fragment is 1, then that fragment must be discarded.

If the length and offset of a fragment are such that the payload length of the packet reassembled from that fragment would exceed 65,535 octets, then that fragment must be discarded.

If a fragment overlaps another fragment already saved for reassembly then the new fragment that overlaps the existing fragment MUST be discarded.

If the first fragment is too small then it is possible that it does not contain the necessary headers for a stateful firewall. Sending small fragments like this has been used as an attack on IP fragmentation. To mitigate this problem, an implementation should ensure that the first fragment contains the headers of the encapsulated packet at least through the transport header.

A GUE node must be able to accept a fragmented packet that, after reassembly and decapsulation, is as large as 1500 octets. This means that the node must configure a reassembly buffer that is at least as large as 1500 octets plus the maximum-sized encapsulation headers that may be inserted during encapsulation. Implementations may find it more convenient and efficient to configure a reassembly buffer size of 2KB which is large enough to accommodate even the largest set of encapsulation headers and provides a natural memory page size boundary.

4.6. Security Considerations

Exploits that have been identified with IP fragmentation are conceptually applicable to GUE fragmentation.

Attacks on GUE fragmentation can be mitigated by:

- o Hardened implementation that applies applicable techniques from implementation of IP fragmentation.
- o Application of GUE security (section 3) or IPsec [RFC4301]. Security mechanisms can prevent spoofing of fragments from unauthorized sources.
- o Implement fragment filter techniques for GUE encapsulation as described in [RFC1858] and [RFC3128].
- o Do not accept data in overlapping segments.
- o Enforce a minimum size for the first fragment.

5. Payload transform option

The payload transform option indicates that the GUE payload has been transformed. Transforming a payload is done by running a function over the data and possibly modifying it (encrypting it for instance). The payload transform option indicates the method used to transform the data so that a decapsulator is able to validate and reverse the transformation to recover the original data. Payload transformations could include encryption, authentication, CRC coverage, and compression. This specification defines a transformation for DTLS.

5.1. Extension field format

The presence of the GUE payload transform option is indicated by the T bit in the GUE header.

The format of Payload Transform Field is:

```

+-----+-----+-----+-----+-----+-----+-----+-----+
|  Type      |  P_C_type  |           Data           |
+-----+-----+-----+-----+-----+-----+-----+

```

The fields of the option are:

Type: Payload Transform Type or Code point. Each payload transform mechanism must have one code point registered in IANA. This document specifies:

0x01: for DTLS [RFC6347]

0x80~0xFF: for private payload transform types

A private payload transform type can be used for experimental purpose or vendor proprietary mechanisms.

P_C_type: Indicates the protocol or control type of the untransformed payload. When payload transform option is present, proto/ctype in the GUE header should set to 59 ("No next header") for a data message and zero for a control message. The IP protocol or control message type of the untransformed payload must be encoded in this field.

The benefit of this rule is to prevent a middle box from inspecting the encrypted payload according to GUE next protocol. The assumption here is that a middle box may understand GUE base header but does not understand GUE option flag definitions.

Data: A field that can be set according to the requirements of each payload transform type. If the specification for a payload transform type does not specify how this field is to be set, then the field MUST be set to zero.

5.2. Usage

The payload transform option provides a mechanism to transform or interpret the payload of a GUE packet. The Type field provides the method used to transform the payload, and the P_C_type field provides the protocol or control message type of the of payload before being transformed. The payload transformation option is generic so that it can have both security related uses (such as DTLS) as well as non security related uses (such as compression, CRC, etc.).

An encapsulator performs payload transformation before transmission, and a decapsulator must perform the reverse transformation before accepting a packet. For example, if an encapsulator transforms a payload by encrypting it, the peer decapsulator must decrypt the payload before accepting the packet. If a decapsulator fails to perform the reverse transformation or cannot validate the transformation it MUST discard the packet and MAY generate an alert to the management system.

5.3. Interaction with other optional extensions

If GUE fragmentation (section 4) is used in concert with the GUE transform option, the transform option processing is performed after

fragmentation at the encapsulator and before reassembly at the decapsulator. If the payload transform changes the size of the data being fragmented this must be taken into account during fragmentation.

If both the security option and the payload transform are used in a GUE packet, an encapsulator must perform the payload transformation first, set the payload transform option in the GUE header, and then create the security option. A decapsulator does processing in reverse-- the security option is processed (GUE header is validated) and then the reverse payload transform is performed.

In order to get flow entropy from the payload, an encapsulator should derive the flow entropy before performing a payload transform.

5.4. DTLS transform

The payload of a GUE packet can be secured using Datagram Transport Layer Security [RFC6347]. An encapsulator would apply DTLS to the GUE payload so that the payload packets are encrypted and the GUE header remains in plaintext. The payload transform option is set to indicate that the payload should be interpreted as a DTLS record.

The payload transform option for DLTS is:

```

+-----+
|      1      | P_C_type |      0      |
+-----+
```

DTLS [RFC6347] provides packet fragmentation capability. To avoid packet fragmentation performed multiple times, a GUE encapsulator SHOULD only perform the packet fragmentation at packet encapsulation process, i.e., not in payload encryption process.

DTLS usage [RFC6347] is limited to a single DTLS session for any specific tunnel encapsulator/decapsulator pair (identified by source and destination IP addresses). Both IP addresses MUST be unicast addresses - multicast traffic is not supported when DTLS is used. A GUE tunnel decapsulator implementation that supports DTLS can establish DTLS session(s) with one or multiple tunnel encapsulators, and likewise a GUE tunnel encapsulator implementation can establish DTLS session(s) with one or multiple decapsulators.

6. Remote checksum offload option

Remote checksum offload is mechanism that provides checksum offload of encapsulated packets using rudimentary offload capabilities found in most Network Interface Card (NIC) devices. Many NIC

implementations can only offload the outer UDP checksum in UDP encapsulation. Remote checksum offload is described in [UDPENCAP].

In remote checksum offload the outer header checksum, that in the outer UDP header, is enabled in packets and, with some additional meta information, a receiver is able to deduce the checksum to be set for an inner encapsulated packet. Effectively this offloads the computation of the inner checksum. Enabling the outer checksum in encapsulation has the additional advantage that it covers more of the packet than the inner checksum including the encapsulation headers.

6.1. Extension field format

The presence of the GUE remote checksum offload option is indicated by the R bit in the GUE header.

The format of remote checksum offload field is:

```

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

```

The fields of the option are:

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

6.2. Usage

6.2.1. 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).
- 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.

6.2.2. 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) Validate the remote checksum option. If checksum start is greater than the length of the packet, then the packet MUST be dropped. If checksum offset is greater than the length of the packet minus two, then the packet MUST be dropped.
- 3) Deduce full checksum for the IP packet. If a NIC is capable of receive checksum offload it will return either the full checksum of the received packet or an indication that the UDP checksum is correct. Either of these methods can be used to deduce the checksum over the IP packet [UDPENCAP].
- 4) From the packet checksum, subtract the checksum computed from the start of the packet (outer IP header) to the offset in the packet indicted by checksum start in the meta data. The result is the deduced checksum to set in the checksum field of the encapsulated transport packet.

In pseudo code:

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

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

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

In pseudo code:

csum_offset: offset of checksum field

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

- 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.

6.3. Security Considerations

Remote checksum offload allows a means to change the GUE payload before being received at a decapsulator. In order to prevent misuse of this mechanism, a decapsulator should apply security checks on the GUE payload only after checksum remote offload has been processed.

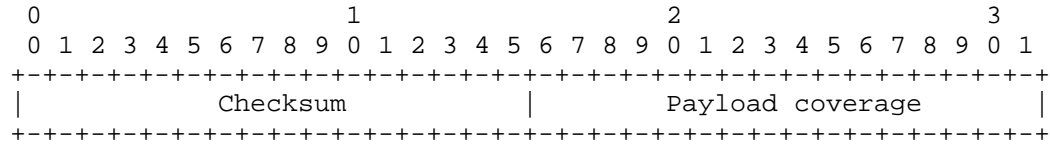
7. Checksum option

The GUE checksum option provides a checksum that covers the GUE header, a GUE pseudo header, and optionally part or all of the GUE payload. The GUE pseudo header includes the corresponding IP addresses as well as the UDP ports of the encapsulating headers. This checksum should provide adequate protection against address corruption in IPv6 when the UDP checksum is zero. Additionally, the GUE checksum provides protection of the GUE header when the UDP checksum is set to zero with either IPv4 or IPv6. In particular, the GUE checksum can provide protection for some sensitive data, such as the virtual network identifier ([I.D.hy-nvo3-gue-4-nvo]), which when corrupted could lead to mis-delivery of a packet to the wrong virtual network.

7.1. Extension field format

The presence of the GUE checksum option is indicated by the K bit in the GUE header.

The format of the checksum extension is:



The fields of the option are:

- o Checksum: Computed checksum value. This checksum covers the GUE header (including fields and private data covered by Hlen), the GUE pseudo header, and optionally all or part of the payload (encapsulated packet).
- o Payload coverage: Number of bytes of payload to cover in the checksum. Zero indicates that the checksum only covers the GUE header and GUE pseudo header. If the value is greater than the encapsulated payload length, the packet must be dropped.

7.2. Requirements

The GUE header checksum should be set on transmit when using a zero UDP checksum with IPv6.

The GUE header checksum should be used when the UDP checksum is zero for IPv4 if the GUE header includes data that when corrupted can lead to misdelivery or other serious consequences, and there is no other mechanism that provides protection (no security field that checks integrity for instance).

The GUE header checksum should not be set when the UDP checksum is non-zero. In this case the UDP checksum provides adequate protection and this avoids convolutions when a packet traverses NAT that does address translation (in that case the UDP checksum is required).

7.3. GUE checksum pseudo header

The GUE pseudo header checksum is included in the GUE checksum to provide protection for the IP and UDP header elements which when corrupted could lead to misdelivery of the GUE packet. The GUE pseudo header checksum is similar to the standard IP pseudo header defined in [RFC0768] and [RFC0793] for IPv4, and in [RFC2460] for IPv6.

The GUE pseudo header for IPv4 is:

```

+-----+
|                                     |
|                               Source Address                               |
|-----+-----+
|                                     |
|                               Destination Address                           |
|-----+-----+
| Source port           | Destination port           |
+-----+-----+

```

The GUE pseudo header for IPv6 is:

```

+-----+
|                                     |
|                               Source Address                               |
|-----+-----+
|                                     |
|                               Destination Address                           |
|-----+-----+
| Source port           | Destination port           |
+-----+-----+

```

Note that the GUE pseudo header does not include payload length or protocol as in the standard IP pseudo headers. The length field is deemed unnecessary because:

- o If the length is corrupted this will usually be detected by a checksum validation failure on the inner packet.
- o Fragmentation of packets in a tunnel should occur on the inner packet before being encapsulated or GUE fragmentation (section 4) may be performed at tunnel ingress. GUE packets are not expected to be fragmented when using IPv6. See RFC6936 for considerations of payload length and IPv6 checksum.
- o A corrupted length field in itself should not lead to misdelivery of a packet.

- o Without the length field, the GUE pseudo header checksum is the same for all packets of flow. This is a useful property for optimizations such as TCP Segment Offload (TSO).

7.4. Usage

The GUE checksum is computed and verified following the standard process for computing the Internet checksum [RFC1071]. Checksum computation may be optimized per the mathematical properties including parallel computation and incremental updates.

7.4.1. Transmitter operation

The procedure for setting the GUE checksum on transmit is:

- 1) Create the GUE header including the checksum and payload coverage fields. The checksum field is initially set to zero.
- 2) Calculate the 1's complement checksum of the GUE header from the start of the GUE header through the its length as indicated in GUE Hlen.
- 3) Calculate the checksum of the GUE pseudo header for IPv4 or IPv6.
- 4) Calculate checksum of payload portion if payload coverage is enabled (payload coverage field is non-zero). If the length of the payload coverage is odd, logically append a single zero byte for the purposes of checksum calculation.
- 5) Add and fold the computed checksums for the GUE header, GUE pseudo header and payload coverage. Set the bitwise not of the result in the GUE checksum field.

7.4.2. Receiver operation

If the GUE checksum option is present, the receiver must validate the checksum before processing any other fields or accepting the packet.

The procedure for verifying the checksum is:

- 1) If the payload coverage length is greater than the length of the encapsulated payload then drop the packet.
- 2) Calculate the checksum of the GUE header from the start of the header to the end as indicated by Hlen.
- 3) Calculate the checksum of the appropriate GUE pseudo header.

- 4) Calculate the checksum of payload if payload coverage is enabled (payload coverage is non-zero). If the length of the payload coverage is odd logically append a single zero byte for the purposes of checksum calculation.
- 5) Sum and fold the computed checksums for the GUE header, GUE pseudo header, and payload coverage. If the result is all 1 bits (-0 in 1's complement arithmetic), the checksum is valid and the packet is accepted; otherwise the checksum is considered invalid and the packet must be dropped.

7.5. Security Considerations

The checksum option is only a mechanism for corruption detection, it is not a security mechanism. To provide integrity checks or authentication of the GUE header, the GUE security option should be used.

8. Processing order of options

Options must be processed in a specific order for both sending and receive.

The order of processing options to send a GUE packet are:

- 1) Set VNID option.
- 2) Fragment if necessary and set fragmentation option. VNID is copied into each fragment. Note that if payload transformation will increase the size of the payload that must be accounted for when deciding how to fragment
- 3) Perform payload transform (potentially on a fragment) and set payload transform option.
- 4) Set Remote checksum offload.
- 5) Set security option.
- 6) Calculate GUE checksum and set checksum option.

On reception the order of actions is reversed.

- 1) Verify GUE checksum.
- 2) Verify security option.

- 3) Adjust packet for remote checksum offload.
- 4) Perform payload transformation (i.e. decrypt payload)
- 5) Perform reassembly.
- 6) Receive on virtual network indicated by VNID.

Note that the relative processing order of private fields is unspecified.

9. Security Considerations

If the integrity and privacy of data packets being transported through GUE is a concern, GUE security option and payload encryption using the the transform option SHOULD be used to remove the concern. If the integrity is the only concern, the tunnel may consider use of GUE security only for optimization. Likewise, if the privacy is the only concern, the tunnel may use GUE encryption function only.

If GUE payload already provides secure mechanism, e.g., the payload is IPsec packets, it is still valuable to consider use of GUE security.

GUE may rely on other secure tunnel mechanisms such as DTLS [RFC6347] over the whole UDP payload for securing the whole GUE packet or IPsec [RFC4301] to achieve the secure transport over an IP network or Internet.

IPsec [RFC4301] was designed as a network security mechanism, and therefore it resides at the network layer. As such, if the tunnel is secured with IPsec, the UDP header would not be visible to intermediate routers in either IPsec tunnel or transport mode. The big drawback here prohibits intermediate routers to perform load balancing based on the flow entropy in UDP header. In addition, this method prohibits any middle box function on the path.

By comparison, DTLS [RFC6347] was designed with application security and can better preserve network and transport layer protocol information than IPsec [RFC4301]. Using DTLS over UDP to secure the GUE tunnel, both GUE header and payload will be encrypted. In order to differentiate plaintext GUE header from encrypted GUE header, the destination port of the UDP header between two must be different, which essentially requires another standard UDP port for GUE with DTLS. The drawback on this method is to prevent a middle box operation to GUE tunnel on the path.

Use of two independent tunnel mechanisms such as GUE and DTLS over

UDP to carry a network protocol over an IP network adds some overlap and complexity. For example, fragmentation will be done twice.

As the result, a GUE tunnel SHOULD use the security mechanisms specified in this document to provide secure transport over an IP network or Internet when it is needed. GUE encapsulation can be used as a secure transport mechanism over an IP network and Internet.

10. IANA Consideration

IANA is requested to assign flags for the extensions defined in this specification. Specifically, an assignment is requested for the V, SEC, F, T, R, and K flags in the "GUE flag-fields" registry (proposed in [I.D.nvo3-gue]).

IANA is requested to set up a registry for the GUE payload transform types. Payload transform types are 8 bit values. New values for control types 1-127 are assigned via Standards Action [RFC5226].

Transform type	Description	Reference
0	Reserved	This document
1	DTLS	This document
2..127	Unassigned	
128..255	User defined	This document

11. References

11.1. 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.
- [RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.
- [RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.
- [I.D.nvo3-gue] T. Herbert, L. Yong, and O. Zia, "Generic UDP

Encapsulation" draft-ietf-nvo3-gue-03

11.2. Informative References

- [RFC6407] Weis, B., Rowles, S., and T. Hardjono, "The Group Domain of Interpretation", RFC 6407, DOI 10.17487/RFC6407, October 2011, <<http://www.rfc-editor.org/info/rfc6407>>.
- [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.
- [RFC4459] MTU and Fragmentation Issues with In-the-Network Tunneling. P. Savola. April 2006.
- [RFC4963] Heffner, J., Mathis, M., and B. Chandler, "IPv4 Reassembly Errors at High Data Rates", RFC 4963, DOI 10.17487/RFC4963, July 2007, <<http://www.rfc-editor.org/info/rfc4963>>.
- [RFC2764] B. Gleeson, A. Lin, J. Heinanen, G. Armitage, A. Malis, "A Framework for IP Based Virtual Private Networks", RFC2764, February 2000.
- [RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.
- [RFC1858] Ziemba, G., Reed, D., and P. Traina, "Security Considerations for IP Fragment Filtering", RFC 1858, October 1995.
- [RFC3128] Miller, I., "Protection Against a Variant of the Tiny Fragment Attack (RFC 1858)", RFC 3128, June 2001.
- [RFC3931] Lau, J., Townsley, W., et al, "Layer Two Tunneling Protocol - Version 3 (L2TPv3)", RFC3931, 1999
- [RFC5925] Touch, J., et al, "The TCP Authentication Option", RFC5925, June 2010.
- [RFC6347] Rescoria, E., Modadugu, N., "Datagram Transport Layer Security Version 1.2", RFC6347, 2012.
- [I.D.hy-nvo3-gue-4-nvo] Yong, L., Herbert, T., "Generic UDP Encapsulation (GUE) for Network Virtualization Overlay"

draft-hy-nvo3-gue-4-nvo-03

[I.D.draft-mathis-frag-harmful] M. Mathis, J. Heffner, and B. Chandler, "Fragmentation Considered Very Harmful"

[I.D.previdi-6man-sr-header] Previdi S. et al, "IPv6 Segment Routing Header (SRH) draft-ietf-6man-segment-routing-header-02

[I.D.templin-aerolink] F. Templin, "Transmission of IP Packets over AERO Links" draft-templin-aerolink-62

[I.D.
[UDPENCAP] T. Herbert, "UDP Encapsulation in Linux",
<http://people.netfilter.org/pablo/netdev0.1/papers/UDP-Encapsulation-in-Linux.pdf>

Authors' Addresses

Tom Herbert
Facebook
1 Hacker Way
Menlo Park, CA
USA

Email: tom@herbertland.com

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

Email: lucy.yong@huawei.com

Fred L. Templin
Boeing Research & Technology
P.O. Box 3707
Seattle, WA 98124
USA

Email: fltemplin@acm.org

INTERNET-DRAFT
Intended Status: Informational
Expires: April 30, 2017

Tom Herbert
Facebook
October 27, 2016

Identifier-locator addressing for IPv6
draft-herbert-nvo3-ila-03

Abstract

This specification describes identifier-locator addressing (ILA) for IPv6. Identifier-locator addressing differentiates between location and identity of a network node. Part of an address expresses the immutable identity of the node, and another part indicates the location of the node which can be dynamic. Identifier-locator addressing can be used to efficiently implement overlay networks for network virtualization as well as solutions for use cases in mobility.

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/lid-abstracts.html>

The list of Internet-Draft Shadow Directories can be accessed at
<http://www.ietf.org/shadow.html>

Copyright and License Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal

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

Table of Contents

1	Introduction	4
1.1	Terminology	4
2	Architectural overview	6
2.1	Addressing	6
2.2	Network topology	6
2.3	Translations and mappings	7
2.4	ILA routing	8
3	Address formats	9
3.1	ILA address format	9
3.2	Locators	9
3.3	Identifiers	9
3.3.1	Checksum neutral-mapping format	10
3.3.2	Identifier types	10
3.3.2.1	Interface identifiers	10
3.3.2.2	Locally unique identifiers	11
3.3.2.3	Virtual networking identifiers for IPv4	11
3.3.2.4	Virtual networking identifiers for IPv6 unicast	12
3.3.2.5	Virtual networking identifiers for IPv6 multicast	13
3.4	Standard identifier representation addresses	14
3.4.1	SIR for locally unique identifiers	15
3.4.2	SIR for virtual addresses	15
3.4.3	SIR domains	16
4	Operation	16
4.1	Identifier to locator mapping	16
4.2	Address translations	16
4.2.1	SIR to ILA address translation	16
4.2.2	ILA to SIR address translation	17
4.3	Virtual networking operation	17
4.3.1	Crossing virtual networks	18
4.3.2	IPv4/IPv6 protocol translation	18
4.4	Transport layer checksums	18
4.4.1	Checksum-neutral mapping	19
4.4.2	Sending an unmodified checksum	20
4.5	Address selection	20
4.6	Duplicate identifier detection	20

4.7 ICMP error handling	21
4.7.1 Handling ICMP errors by ILA capable hosts	21
4.7.2 Handling ICMP errors by non-ILA capable hosts	21
4.8 Multicast	22
5 Motivation for ILA	22
5.1 Use cases	22
5.1.1 Multi-tenant virtualization	22
5.1.2 Datacenter virtualization	23
5.1.3 Device mobility	23
5.2 Alternative methods	24
5.2.1 ILNP	24
5.2.2 Flow label as virtual network identifier	24
5.2.3 Extension headers	25
5.2.4 Encapsulation techniques	25
6 IANA Considerations	26
7 References	26
7.1 Normative References	26
7.2 Informative References	26
8 Acknowledgments	27
Appendix A: Communication scenarios	28
A.1 Terminology for scenario descriptions	28
A.2 Identifier objects	29
A.3 Reference network for scenarios	29
A.4 Scenario 1: Object to task	30
A.5 Scenario 2: Object to Internet	30
A.6 Scenario 3: Internet to object	30
A.7 Scenario 4: Tenant system to service	31
A.8 Scenario 5: Object to tenant system	31
A.9 Scenario 6: Tenant system to Internet	32
A.10 Scenario 7: Internet to tenant system	32
A.11 Scenario 8: IPv4 tenant system to object	32
A.12 Tenant to tenant system in the same virtual network	33
A.12.1 Scenario 9: TS to TS in the same VN using IPV6	33
A.12.2 Scenario 10: TS to TS in same VN using IPv4	33
A.13 Tenant system to tenant system in different virtual networks	33
A.13.1 Scenario 11: TS to TS in different VNs using IPV6	33
A.13.2 Scenario 12: TS to TS in different VNs using IPv4	34
A.13.3 Scenario 13: IPv4 TS to IPv6 TS in different VNs	34
Appendix B: unique identifier generation	35
B.1 Globally unique identifiers method	35
B.2 Universally Unique Identifiers method	35
Appendix C: Datacenter task virtualization	36
C.1 Address per task	36
C.2 Job scheduling	36
C.3 Task migration	37
C.3.1 Address migration	37
C.3.2 Connection migration	38

1 Introduction

This specification describes the address formats, protocol operation, and communication scenarios of identifier-locator addressing (ILA). In identifier-locator addressing, an IPv6 address is split into a locator and an identifier component. The locator indicates the topological location in the network for a node, and the identifier indicates the node's identity which refers to the logical or virtual node in communications. Locators are routable within a network, but identifiers typically are not. An application addresses a peer destination by identifier. Identifiers are mapped to locators for transit in the network. The on-the-wire address is composed of a locator and an identifier: the locator is sufficient to route the packet to a physical host, and the identifier allows the receiving host to translate and forward the packet to the addressed application.

With identifier-locator addressing network virtualization and addressing for mobility can be implemented in an IPv6 network without any additional encapsulation headers. Packets sent with identifier-locator addresses look like plain unencapsulated packets (e.g. TCP/IP packets). This method is transparent to the network, so protocol specific mechanisms in network hardware work seamlessly. These mechanisms include hash calculation for ECMP, NIC large segment offload, checksum offload, etc.

Many of the concepts for ILA are adapted from Identifier-Locator Network Protocol (ILNP) ([RFC6740], [RFC6741]) which defines a protocol and operations model for identifier-locator addressing in IPv6.

Section 5 provides a motivation for ILA and comparison of ILA with alternative methods that achieve similar functionality.

1.1 Terminology

ILA	Identifier-locator addressing.
ILA router	A network node that performs ILA translation and forwarding of translated packets.
ILA host	An end host that is capable of performing ILA translations on transmit or receive.
ILA node	A network node capable of performing ILA translations. This can be an ILA router or ILA host.
Locator	A network prefix that routes to a physical host.

Locators provide the topological location of an addressed node. In ILA locators are a sixty-four bit prefixes.

Identifier A number that identifies an addressable node in the network independent of its location. ILA identifiers are sixty-four bit values.

ILA address An IPv6 address composed of a locator (upper sixty-four bits) and an identifier (low order sixty-four bits).

SIR Standard identifier representation.

SIR prefix A sixty-four bit network prefix used to identify a SIR address.

SIR address An IPv6 address composed of a SIR prefix (upper sixty-four bits) and an identifier (lower sixty-four bits). SIR addresses are visible to applications and provide a means to address nodes independent of their location.

SIR domain A unique identifier namespace defined by a SIR prefix. Each SIR prefix defines a SIR domain.

ILA translation The process of translating the upper sixty-four bits of an IPv6 address. Translations may be from a SIR prefix to a locator or a locator to a SIR prefix.

Virtual address An IPv6 or IPv4 address that resides in the address space of a virtual network. Such addresses may be translated to SIR addresses as an external representation of the address outside of the virtual network, or they may be translated to ILA addresses for transit over an underlay network.

Topological address An address that refers to a non-virtual node in a network topology. These address physical hosts in a network.

2 Architectural overview

Identifier-locator addressing allows a data plane method to implement network virtualization without encapsulation and its related overheads. The service ILA provides is effectively layer 3 over layer 3 network virtualization (IPv4 or IPv6 over IPv6).

2.1 Addressing

ILA performs translations on IPv6 address. There are two types of addresses introduced for ILA: ILA addresses and SIR addresses.

ILA addresses are IPv6 addresses that are composed of a locator (upper sixty-four bits) and an identifier (low order sixty-four bits). The identifier serves as the logical addresses of a node, and the locator indicates the location of the node on the network.

A SIR address (standard identifier representation) is an IPv6 address that contains an identifier and an application visible SIR prefix. SIR addresses are visible to the application and can be used as connection endpoints. When a packet is sent to a SIR address, an ILA router or host overwrites the SIR prefix with a locator corresponding to the identifier. When a peer ILA node receives the packet, the locator is overwritten with the original SIR prefix before delivery to the application. In this manner applications only see SIR addresses, they do not have visibility into ILA addresses.

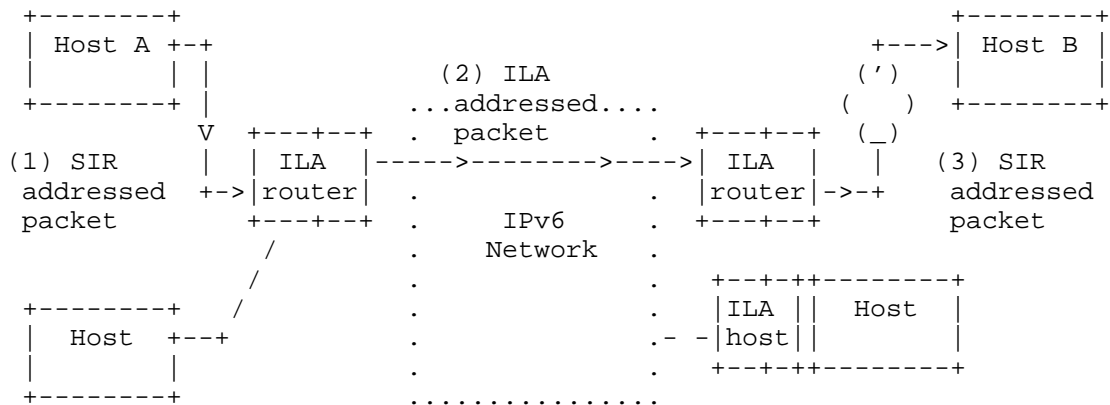
ILA translations can transform addresses from one type to another. In network virtualization virtual addresses can be translated into ILA and SIR addresses, and conversely ILA and SIR addresses can be translated to virtual addresses.

2.2 Network topology

ILA nodes are nodes in the network that perform ILA translations. An ILA router is a node that performs ILA address translation and packet forwarding to implement overlay network functionality. ILA routers perform translations on packets sent by end nodes for transport across an underlay network. Packets received by ILA routers on the underlay network have their addresses reversed translated for reception at an end node. An ILA host is an end node that implements ILA functionality for transmitting or receiving packets.

ILA nodes are responsible for transit of packets over an underlay network. On ingress to an ILA node (host or router) the virtual or SIR address of a destination is translated to an ILA address. At the a peer ILA node, the reverse translation is performed before handing packets to an application.

The figure below provides an example topology using ILA. ILA translations performed in one direction between Host A and Host B are denoted. Host A sends a packet with a destination SIR address (step (1)). An ILA router in the path translates the SIR address to an ILA address with a locator set to Host B, referring to the location of the node indicated by the identifier in the SIR address. The packet is forwarded over the network and delivered to a peer ILA node (step 2). The peer ILA node, in this case another ILA router, translates the destination address back to a SIR address and forwards to the final destination (step 3).



2.3 Translations and mappings

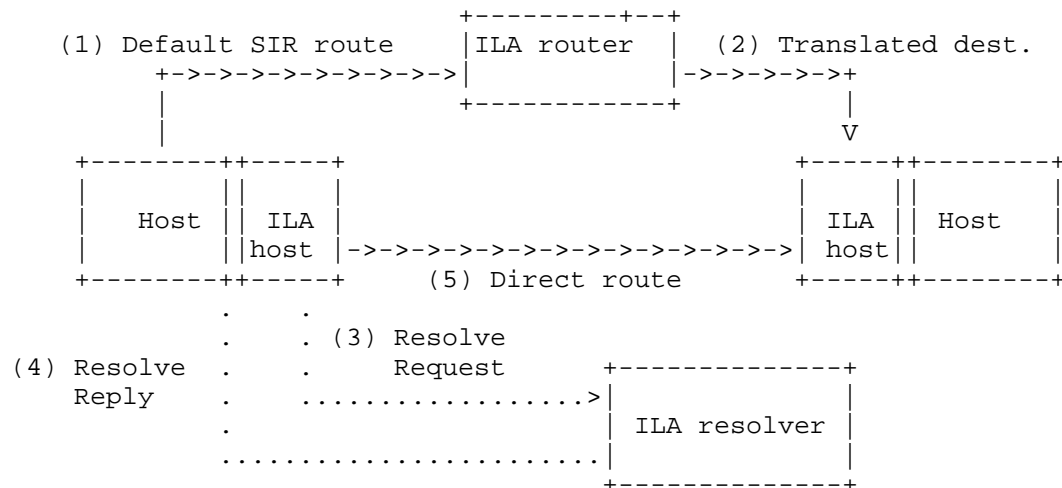
Address translation is the mechanism employed by ILA. Logical or virtual addresses are translated to topological IPv6 addresses for transport to the proper destination. Translation occurs in the upper sixty-four bits of an address, the low order sixty-four bits contains an identifier that is immutable and is not used to route a packet.

Each ILA node maintains a mapping table. This table maps identifiers to locators. The mappings are dynamic as nodes with identifiers can be created, destroyed, or migrated between physical hosts. Mappings are propagated amongst ILA routers or hosts in a network using mapping propagation protocols (mapping propagation protocols will be described in other specifications).

Identifiers are not statically bound to a host on the network, and in fact their binding (or location) may change. This is the basis for network virtualization and address migration. An identifier is mapped to a locator at any given time, and a set of identifier to locator mappings is propagated throughout a network to allow communications. The mappings are kept synchronized so that if an identifier migrates to a new physical host, its identifier to locator mapping is updated.

2.4 ILA routing

ILA is intended to be sufficiently lightweight so that all the hosts in a network could potentially send and receive ILA addressed packets. In order to scale this model and allow for hosts that do not participate in ILA, a routing topology may be applied. A simple routing topology is illustrated below.



An ILA router can be addressed by an "anycast" SIR prefix so that it receives packets sent on the network with SIR addresses. When an ILA router receives a SIR addressed packet (step (1) in the diagram) it will perform the ILA translation and send the ILA addressed packet to the destination ILA node (step (2)).

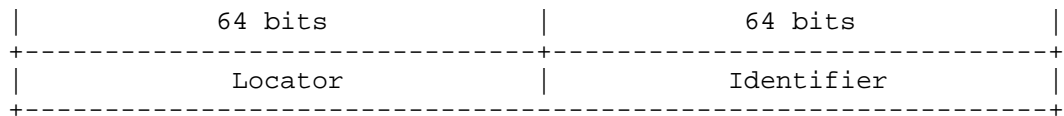
If a sending host is ILA capable the triangular routing can be eliminated by performing an ILA resolution protocol. This entails the host sending an ILA resolve request that specifies the SIR address to resolve (step (3) in the figure). An ILA resolver can respond to a resolver request with the identifier to locator mapping (step (4)). Subsequently, the ILA host can perform ILA translation and send directly to the destination specified in the locator (step (5) in the figure). The ILA resolution protocol will be specified in a companion document.

In this model an ILA host maintains a cache of identifier mappings for identifiers that it is currently communicating with. ILA routers are expected to maintain a complete list of identifier to locator mappings within the SIR domains that they service.

3 Address formats

3.1 ILA address format

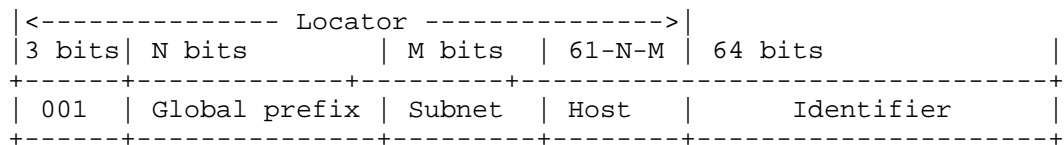
An ILA address is composed of a locator and an identifier where each occupies sixty-four bits (similar to the encoding in ILNP [RFC6741]).



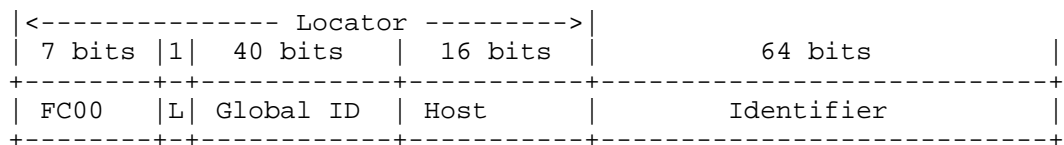
3.2 Locators

Locators are routable network address prefixes that create topological addresses for physical hosts within the network. They may be assigned from a global address block [RFC3587], or be based on unique local IPv6 unicast addresses as described in [RFC4193].

The format of an ILA address with a global unicast locator is:

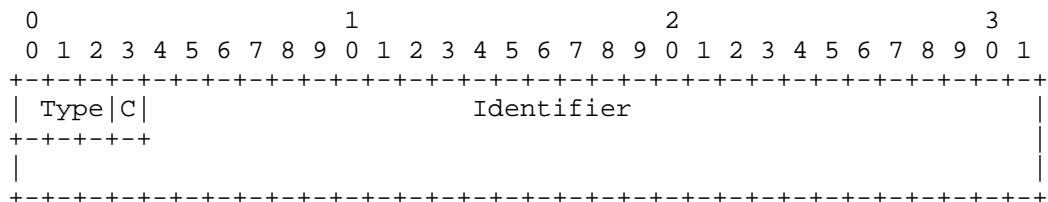


The format of an ILA address with a unique local IPv6 unicast locator is:



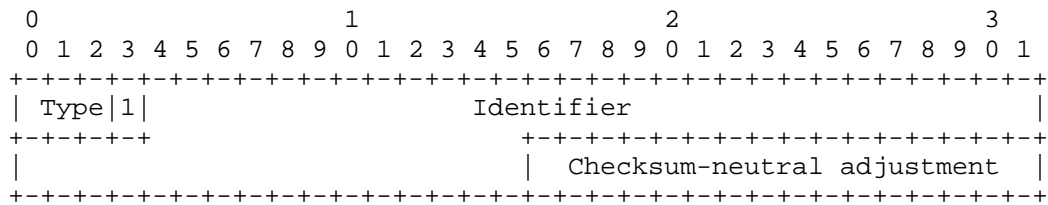
3.3 Identifiers

The format of an ILA identifier is:



- o Type: Type of the identifier (see section 3.3.2).
- o C: The C-bit. This indicates that checksum-neutral mapping applied (see section 3.3.1).
- o Identifier: Identifier value.

If the C-bit is set the low order sixteen bits of an identifier contain the adjustment for checksum-neutral mapping (see section 4.4.1 for description of checksum-neutral mapping). The format of an identifier with checksum neutral mapping is:



Identifier types allow standard encodings for common uses of identifiers. Defined identifier types are:

- ### 3.3.2.1 Interface identifiers

[Page 10]

The format of an ILA interface identifier address is:

	64 bits		3 bits		1		60 bits	
+	-----	+	-----	+	-----	+	-----	+
	Prefix		0x0		0		IID	
+	-----	+	-----	+	-----	+	-----	+

3.3.2.2 Locally unique identifiers

Locally unique identifiers (LUI) can be created for various addressable objects within a network. These identifiers are in a flat sixty bit space and must be unique within a SIR domain (unique within a site for instance). To simplify administration, hierarchical allocation of locally unique identifiers may be performed. The format of an ILA address with locally unique identifiers is:

	64 bits		3 bits		1		60 bits	
+	-----	+	-----	+	-----	+	-----	+
	Locator		0x1		C		Locally unique ident.	
+	-----	+	-----	+	-----	+	-----	+

The figure below illustrates the translation from SIR address to an ILA address as would be performed when a node sends to a SIR address. Note the low order 16 bits of the identifier may be modified as the checksum-neutral adjustment. The reverse translation of ILA address to SIR address is symmetric.

+	-----	+	-----	+	-----	+	-----	+
	SIR prefix		0x1		0		Identifier	
+	-----	+	-----	+	-----	+	-----	+
	SIR prefix to locator		C-bit if needed					
	V		V				V	
+	-----	+	-----	+	-----	+	-----	+
	Locator		0x1		C		Identifier	
+	-----	+	-----	+	-----	+	-----	+

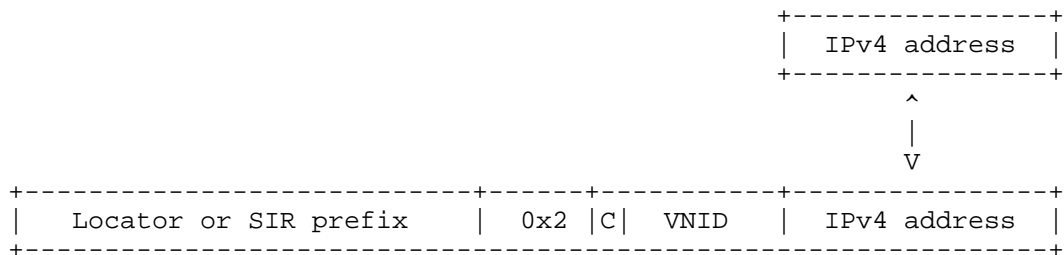
3.3.2.3 Virtual networking identifiers for IPv4

This type defines a format for encoding an IPv4 virtual address and virtual network identifier within an identifier. The format of an ILA address for IPv4 virtual networking is:

	64 bits		3 bits		1		28 bits		32 bits	
+	-----	+	-----	+	-----	+	-----	+	-----	+
	Locator		0x2		C		VNID		VADDR	
+	-----	+	-----	+	-----	+	-----	+	-----	+

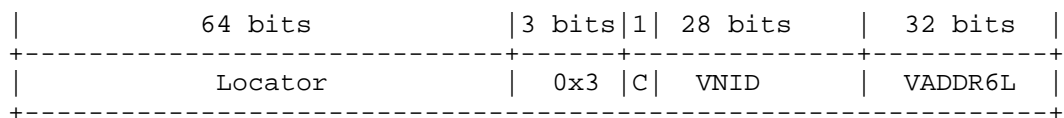
VNID is a virtual network identifier and VADDR is a virtual address within the virtual network indicated by the VNID. The VADDR can be an IPv4 unicast or multicast address, and may often be in a private address space (i.e. [RFC1918]) used in the virtual network.

Translating a virtual IPv4 address into an ILA or SIR address and the reverse translation are straight forward. Note that the low order 16 bits of the IPv6 address may be modified as the checksum-neutral adjustment and that this translation implies protocol translation when sending IPv4 packets over an ILA IPv6 network.



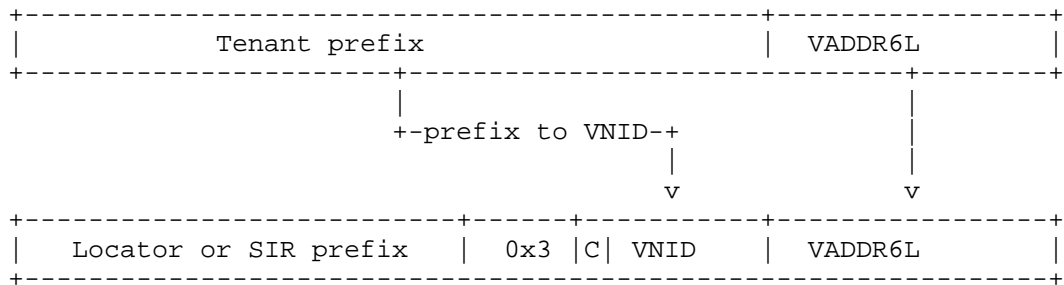
3.3.2.4 Virtual networking identifiers for IPv6 unicast

In this format, a virtual network identifier and virtual IPv6 unicast address are encoded within an identifier. To facilitate encoding of virtual addresses, there is a unique mapping between a VNID and a ninety-six bit prefix of the virtual address. The format an IPv6 unicast encoding with VNID in an ILA address is:

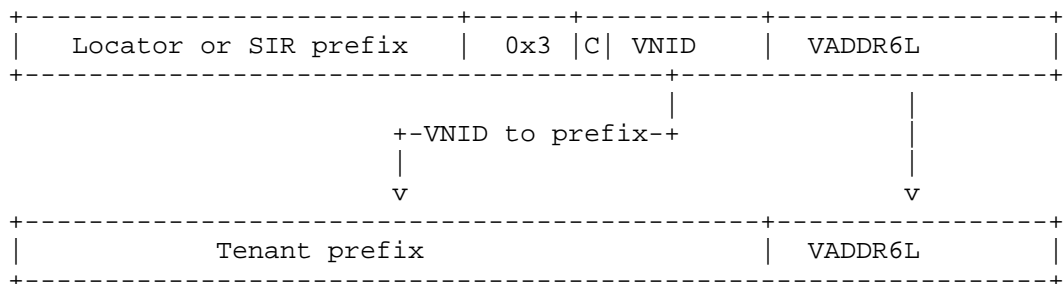


VADDR6L contains the low order 32 bits of the IPv6 virtual address. The upper 96 bits of the virtual address are inferred from the VNID to prefix mapping. Note that for ILA translations the low order sixteen of the VADDR6L may be modified for checksum-neutral adjustment.

The figure below illustrates encoding a tenant IPv6 virtual unicast address into a ILA or SIR address.

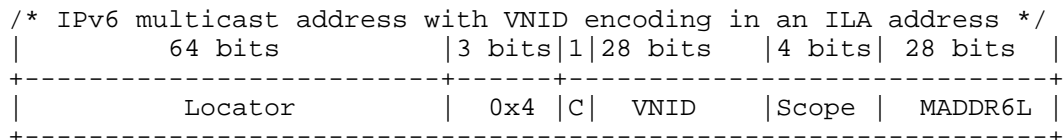


This encoding is reversible, given an ILA address, the virtual address visible to the tenant can be deduced:



3.3.2.5 Virtual networking identifiers for IPv6 multicast

In this format, a virtual network identifier and virtual IPv6 multicast address are encoded within an identifier.



This format encodes an IPv6 multicast address in an identifier. The scope indicates multicast address scope as defined in [RFC7346]. MADDR6L is the low order 28 bits of the multicast address. The full multicast address is thus:

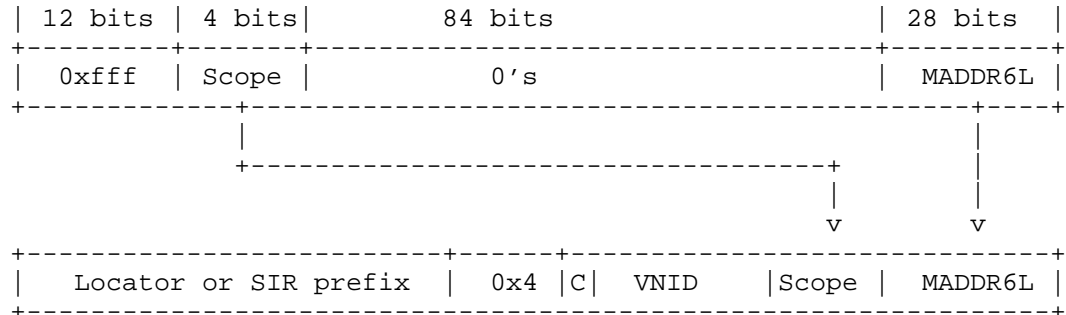
ff0<Scope>::<MADDR6L high 12 bits>:<MADDR6L low 16 bits>

And so can encode multicast addresses of the form:

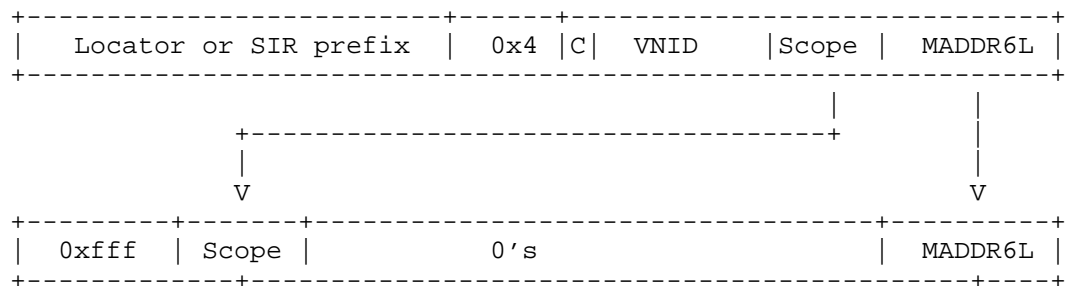
ff0X::0 to ff0X::0fff:ffff

The figure below illustrates encoding a tenant IPv6 virtual multicast

address in an ILA or SIR address. Note that low order sixteen bits of MADDR6L may be modified to be the checksum-neutral adjustment.



This translation is reversible:



3.4 Standard identifier representation addresses

An identifier identifies objects or nodes in a network. For instance, an identifier may refer to a specific host, virtual machine, or tenant system. When a host initiates a connection or sends a packet, it uses the identifier to indicate the peer endpoint of the communication. The endpoints of an established connection context also referenced by identifiers. It is only when the packet is actually being sent over a network that the locator for the identifier needs to be resolved.

In order to maintain compatibility with existing networking stacks and applications, identifiers are encoded in IPv6 addresses using a standard identifier representation (SIR) address. A SIR address is a combination of a prefix which occupies what would be the locator portion of an ILA address, and the identifier in its usual location. The format of a SIR address is:

	64 bits	3 bits 1	60 bits	
+-----+		+-----+		+-----+
	SIR prefix	Type 0	Identifier	
+-----+		+-----+		+-----+

The C-bit (checksum-neutral mapping) MUST be zero for a SIR address. Type may be any identifier type except zero (interface identifiers)

A SIR prefix may be site-local, or globally routable. A globally routable SIR prefix facilitates connectivity between hosts on the Internet and ILA nodes. A gateway between a site's network and the Internet can translate between SIR prefix and locator for an identifier. A network may have multiple SIR prefixes where each prefix defines a unique identifier space.

Locators MUST only be associated with one SIR prefix. This ensures that if a translation from a SIR address to an ILA address is performed when sending a packet, the reverse translation at the receiver yields the same SIR address that was seen at the transmitter. This also ensures that a reverse checksum-neutral mapping can be performed at a receiver to restore the addresses that were included in a pseudo header for setting a transport checksum.

A standard identifier representation address can be used as the externally visible address for a node. This can be used throughout the network, returned in DNS AAAA records [RFC3363], used in logging, etc. An application can use a SIR address without knowledge that it encodes an identifier.

3.4.1 SIR for locally unique identifiers

The SIR address for a locally unique identifier has format:

	64 bits	3 bits 1	60 bits	
+-----+		+-----+		+-----+
	SIR prefix	0x1 0	Locally unique ident.	
+-----+		+-----+		+-----+

3.4.2 SIR for virtual addresses

A virtual address can be encoded using the standard identifier representation. For example, the SIR address for an IPv6 virtual address may be:

	64 bits	3 bits 1	28 bits		32 bits	
+-----+		+-----+		+-----+		+-----+
	SIR prefix	0x3 0	VNID		VADDRL6	
+-----+		+-----+		+-----+		+-----+

Note that this allows three representations of the same address in the network: as a virtual address, a SIR address, and an ILA address.

3.4.3 SIR domains

Each SIR prefix defines a SIR domain. A SIR domain is a unique name space for identifiers within a domain. The full identity of a node is thus determined by an identifier and SIR domain (SIR prefix). Locators MUST map to only one SIR domain in order to ensure that translation from a locator to SIR prefix is unambiguous.

4 Operation

This section describes operation methods for using identifier-locator addressing.

4.1 Identifier to locator mapping

An application initiates a communication or flow using a SIR address or virtual address for a destination. In order to send a packet on the network, the destination address is translated by an ILA router or an ILA host in the path. An ILA node maintains a list of mappings from identifier to locator to perform this translation.

The mechanisms of propagating and maintaining identifier to locator mappings are outside the scope of this document.

4.2 Address translations

With ILA, address translation is performed to convert SIR addresses to ILA addresses, and ILA addresses to SIR addresses. Translation is usually done on a destination address as a form of source routing, however translation on source virtual addresses to SIR addresses can also be done to support some network virtualization scenarios (see appendix A.7 for example).

4.2.1 SIR to ILA address translation

When translating a SIR address to an ILA address the SIR prefix in the address is overridden with a locator, and checksum neutral mapping may be performed. Since this operation is potentially done for every packet the process should be very efficient (particularly the lookup and checksum processing operations).

The typical steps to transmit a packet using ILA are:

- 1) Host stack creates a packet with source address set to a local address (possibly a SIR address) for the local identity, and

the destination address is set to the SIR address or virtual address for the peer. The peer address may have been discovered through DNS or other means.

- 2) An ILA router or host translates the packet to use the locator. If the original destination address is a SIR address then the SIR prefix is overwritten with the locator. If the original packet is a virtually addressed tenant packet then the virtual address is translated per section 3.3.2. The locator is discovered by a lookup in the locator to identifier mappings.
- 3) The ILA node performs checksum-neutral mapping if configured for that (section 4.4.1).
- 4) Packet is forwarded on the wire. The network routes the packet to the host indicated by the locator.

4.2.2 ILA to SIR address translation

When a destination node (ILA router or end host) receives an ILA addressed packet, the ILA address **MUST** be translated back to a SIR address (or tenant address) before upper layer processing.

The steps of receive processing are:

- 1) Packet is received. The destination locator is verified to match a locator assigned to the host.
- 2) A lookup is performed on the destination identifier to find if it addresses a local identifier. If match is found, either the locator is overwritten with SIR prefix (for locally unique identifier type) or the address is translated back to a tenant virtual address as shown in appendix A.7.
- 3) Perform reverse checksum-neutral mapping if C-bit is set (section 4.4.1).
- 4) Perform any optional policy checks; for instance that the source may send a packet to the destination address, that packet is not illegitimately crossing virtual networks, etc.
- 5) Forward packet to application processing.

4.3 Virtual networking operation

When using ILA with virtual networking identifiers, address translation is performed to convert tenant virtual network and virtual addresses to ILA addresses, and ILA addresses back to a

virtual network and tenant's virtual addresses. Translation may occur on either source address, destination address, or both (see scenarios for virtual networking in Appendix A). Address translation is performed similar to the SIR translation cases described above.

4.3.1 Crossing virtual networks

With explicit configuration, virtual network hosts may communicate directly with virtual hosts in another virtual network by using SIR addresses for virtualization in both the source and destination addresses. This might be done to allow services in one virtual network to be accessed from another (by prior agreement between tenants). See appendix A.13 for example of ILA addressing for such a scenario.

4.3.2 IPv4/IPv6 protocol translation

An IPv4 tenant may send a packet that is converted to an IPv6 packet with ILA addresses. Similarly, an IPv6 packet with ILA addresses may be converted to an IPv4 packet to be received by an IPv4-only tenant. These are IPv4/IPv6 stateless protocol translations as described in [RFC6144] and [RFC6145]. See appendix A.12 for a description of these scenarios.

4.4 Transport layer checksums

Packets undergoing ILA translation may encapsulate transport layer checksums (e.g. TCP or UDP) that include a pseudo header that is affected by the translation.

ILA provides two alternatives to deal with this:

- o Perform a checksum-neutral mapping to ensure that an encapsulated transport layer checksum is kept correct on the wire.
- o Send the checksum as-is, that is send the checksum value based on the pseudo header before translation.

Some intermediate devices that are not the actual end point of a transport protocol may attempt to validate transport layer checksums. In particular, many Network Interface Cards (NICs) have offload capabilities to validate transport layer checksums (including any pseudo header) and return a result of validation to the host. Typically, these devices will not drop packets with bad checksums, they just pass a result to the host. Checksum offload is a performance benefit, so if packets have incorrect checksums on the wire this benefit is lost. With this incentive, applying a checksum-

neutral mapping is the recommended alternative. If it is known that the addresses of a packet are not included in a transport checksum, for instance a GRE packet is being encapsulated, then a source may choose not to perform checksum-neutral mapping.

4.4.1 Checksum-neutral mapping

When a change is made to one of the IP header fields in the IPv6 pseudo-header checksum (such as one of the IP addresses), the checksum field in the transport layer header may become invalid. Fortunately, an incremental change in the area covered by the Internet standard checksum [RFC1071] will result in a well-defined change to the checksum value [RFC1624]. So, a checksum change caused by modifying part of the area covered by the checksum can be corrected by making a complementary change to a different 16-bit field covered by the same checksum.

ILA can perform a checksum-neutral mapping when a SIR prefix or virtual address is translated to a locator in an IPv6 address, and performs the reverse mapping when translating a locator back to a SIR prefix or virtual address. The low order sixteen bits of the identifier contain the checksum adjustment value for ILA.

On transmission, the translation process is:

- 1) Compute the one's complement difference between the SIR prefix and the locator. Fold this value to 16 bits (add-with-carry four 16-bit words of the difference).
- 2) Add-with-carry the bit-wise not of the 0x1000 (i.e. 0xffff) to the value from #1. This compensates the checksum for setting the C-bit.
- 3) Add-with-carry the value from #2 to the low order sixteen bits of the identifier.
- 4) Set the resultant value from #3 in the low order sixteen bits of the identifier and set the C-bit.

Note that the "adjustment" (the 16-bit value set in the identifier in set #3) is fixed for a given SIR to locator mapping, so the adjustment value can be saved in an associated data structure for a mapping to avoid computing it for each translation.

On reception of an ILA addressed packet, if the C-bit is set in an ILA address:

- 1) Compute the one's complement difference between the locator in

the address and the SIR prefix that the locator is being translated to. Fold this value to 16 bits (add-with-carry four 16-bit words of the difference).

- 2) Add-with-carry 0x1000 to the value from #1. This compensates the checksum for clearing the C-bit.
- 3) Add-with-carry the value from #2 to the low order sixteen bits of the identifier.
- 4) Set the resultant value from #3 in the low order sixteen bits of the identifier and clear the C-bit. This restores the original identifier sent in the packet.

4.4.2 Sending an unmodified checksum

When sending an unmodified checksum, the checksum is incorrect as viewed in the packet on the wire. At the receiver, ILA translation of the destination ILA address back to the SIR address occurs before transport layer processing. This ensures that the checksum can be verified when processing the transport layer header containing the checksum. Intermediate devices are not expected to drop packets due to a bad transport layer checksum.

4.5 Address selection

There may be multiple possibilities for creating either a source or destination address. A node may be associated with more than one identifier, and there may be multiple locators for a particular identifier. The choice of locator or identifier is implementation or configuration specific. The selection of an identifier occurs at flow creation and must be invariant for the duration of the flow. Locator selection must be done at least once per flow, and the locator associated with the destination of a flow may change during the lifetime of the flow (for instance in the case of a migrating connection it will change). ILA address selection should follow specifications in Default Address Selection for Internet Protocol Version 6 (IPv6) [RFC6724].

4.6 Duplicate identifier detection

As part of implementing the locator to identifier mapping, duplicate identifier detection should be implemented in a centralized control plane. A registry of identifiers could be maintained (possibly in association the identifier to locator mapping database). When a node creates an identifier it registers the identifier, and when the identifier is no longer in use (e.g. task completes) the identifier is unregistered. The control plane should be able to detect a

registration attempt for an existing identifier and deny the request.

4.7 ICMP error handling

A packet that contains an ILA address may cause ICMP errors within the network. In this case the ICMP data contains an IP header with an ILA address. ICMP messages are sent back to the source address in the packet. Upon receiving an ICMP error the host will process it differently depending on whether it is ILA capable.

4.7.1 Handling ICMP errors by ILA capable hosts

If a host is ILA capable it can attempt to reverse translate the ILA address in the destination of a header in the ICMP data back to a SIR address that was originally used to transmit the packet. The steps are:

- 1) Assume that the upper sixty-four bits of the destination address in the ICMP data is a locator. Try match these bits back to a SIR address. If the host is only in one SIR domain, then the mapping to SIR address is implicit. If the host is in multiple domains then a locator to SIR addresses table can be maintained for this lookup.
- 2) If the identifier is marked with checksum-neutral mapping, undo the checksum-neutral using the SIR address found in #1. The resulting identifier address is potentially the original address used to send the packet.
- 3) Lookup the identifier in the identifier to locator mapping table. If an entry is found compare the locator in the entry to the locator (upper sixty-four bits) of the destination address in the IP header of the ICMP data. If these match then proceed to next step.
- 4) Overwrite the upper sixty-four bits of the destination address in the ICMP data with the found SIR address and overwrite the low order sixty-four bits with the found identifier (the result of undoing checksum-neutral mapping). The resulting address should be the original SIR address used in sending. The ICMP error packet can then be received by the stack for further processing.

4.7.2 Handling ICMP errors by non-ILA capable hosts

A non-ILA capable host may receive an ICMP error generated by the network that contains an ILA address in an IP header contained in the ICMP data. This would happen in the case that an ILA router performed

translation on a packet the host sent and that packet subsequently generated an ICMP error. In this case the host receiving the error message will attempt to find the connection state corresponding to the packet in headers the ICMP data. Since the host is unaware of ILA the lookup for connection state should fail. Because the host cannot recover the original addresses it used to send the packet, it won't be able any to derive any useful information about the original destination of the packet that it sent.

If packets for a flow are always routed through an ILA router in both directions, for example ILA routers are coincident with edge routes in a network, then ICMP errors could be intercepted by an intermediate node which could translate the destination addresses in ICMP data back to the original SIR addresses. A receiving host would then see the destination address in the packet of the ICMP data to be that it used to transmit the original packet.

4.8 Multicast

ILA is generally not intended for use with multicast. In the case of multicast, routing of packets is based on the source address. Neither the SIR address nor an ILA address is suitable for use as a source address in a multicast packet. A SIR address is unroutable and hence would make a multicast packet unroutable if used as a source address. Using an ILA address as the source address makes the multicast packet routable, but this exposes ILA address to applications which is especially problematic on a multicast receiver that doesn't support ILA.

If all multicast receivers are known to support ILA, a local locator address may be used in the source address of the multicast packet. In this case, each receiver will translate the source address from an ILA address to a SIR address before delivering packets to an application.

5 Motivation for ILA

5.1 Use cases

5.1.1 Multi-tenant virtualization

In multi-tenant virtualization overlay networks are established for tenants to provide virtual networks. Each tenant may have one or more virtual networks and a tenant's nodes are assigned virtual addresses within virtual networks. Identifier-locator addressing may be used as an alternative to traditional network virtualization encapsulation protocols used to create overlay networks (e.g. VXLAN [RFC7348]). Section 5.2,4 describes the advantages of using ILA in lieu of

encapsulation protocols.

Tenant systems (e.g. VMs) run on physical hosts and may migrate to different hosts. A tenant system is identified by a virtual address and virtual networking identifier of a corresponding virtual network. ILA can encode the virtual address and a virtual networking identifier in an ILA identifier. Each identifier is mapped to a locator that indicates the current host where the tenant system resides. Nodes that send to the tenant system set the locator per the mapping. When a tenant system migrates its identifier to locator mapping is updated and communicating nodes will use the new mapping.

5.1.2 Datacenter virtualization

Datacenter virtualization virtualizes networking resources. Various objects within a datacenter can be assigned addresses and serve as logical endpoints of communication. A large address space, for example that of IPv6, allows addressing to be used beyond the traditional concepts of host based addressing. Addressed objects can include tasks, virtual IP addresses (VIPs), pieces of content, disk blocks, etc. Each object has a location which is given by the host on which an object resides. Some objects may be migratable between hosts such that their location changes over time.

Objects are identified by a unique identifier within a namespace for the datacenter (appendix B discusses methods to create unique identifiers for ILA). Each identifier is mapped to a locator that indicates the current host where the object resides. Nodes that send to an object set the locator per the mapping. When an object migrates its identifier to locator mapping is updated and communicating nodes will use the new mapping.

A datacenter object of particular interest is tasks, units of execution for applications. The goal of virtualizing tasks is to maximize resource efficiency and job scheduling. Tasks share many properties of tenant systems, however they are finer grained objects, may have a shorter lifetimes, and are likely created in greater numbers. Appendix C provides more detail and motivation for virtualizing tasks using ILA.

5.1.3 Device mobility

ILA may be applied as a solution for mobile devices. These are devices, smart phones for instance, that physically move between different networks. The goal of mobility is to provide a seamless transition when a device moves from one network to another.

Each mobile device is identified by unique identifier within some

provider domain. ILA encodes the identifier for the device in an ILA identifier. Each identifier is mapped to a locator that indicates the current network or point of attachment for the device. Nodes that send to the device set the locator per the mapping. When a mobile device moves between networks its identifier to locator mapping is updated and communicating nodes will use the new mapping.

5.2 Alternative methods

This section discusses the merits of alternative solution that have been proposed to provide network virtualization or mobility in IPv6.

5.2.1 ILNP

ILNP splits an address into a locator and identifier in the same manner as ILA. ILNP has characteristics, not present in ILA, that prevent it from being a practical solution:

- o ILNP requires that transport layer protocol implementations must be modified to work over ILNP.
- o ILNP can only be implemented in end hosts, not within the network. This essentially requires that all end hosts need to be modified to participate in mobility.
- o ILNP employs IPv6 extension headers which are mostly considered non-deployable. ILA does not use these.
- o Core support for ILA is in upstream Linux, to date there is no publicly available source code for ILNP.
- o ILNP involves DNS to distribute mapping information, ILA assumes mapping information is not part of naming.

5.2.2 Flow label as virtual network identifier

The IPv6 flow label could conceptually be used as a 20-bit virtual network identifier in order to indicate a packet is sent on an overlay network. In this model the addresses may be virtual addresses within the specified virtual network. Presumably, the tuple of flow-label and addresses could be used by switches to forward virtually addressed packets.

This approach has some issues:

- o Forwarding virtual packets to their physical location would require specialized switch support.

- o The flow label is only twenty bits, this is too small to be a discriminator in forwarding a virtual packet to a specific destination. Conceptually, the flow label might be used in a type of label switching to solve that.
- o The flow label is not considered immutable in transit, intermediate devices may change it.
- o The flow label is not part of the pseudo header for transport checksum calculation, so it is not covered by any transport (or other) checksums.

5.2.3 Extension headers

To accomplish network virtualization an extension header, as a destination or routing option, could be used that contains the virtual destination address of a packet. The destination address in the IPv6 header would be the topological address for the location of the virtual node. Conceivably, segment routing could be used to implement network virtualization in this manner.

This technique has some issues:

- o Intermediate devices must not insert extension headers [RFC2460bis].
- o Extension headers introduce additional packet overhead which may impact performance.
- o Extension headers are not covered by transport checksums (as the addresses would be) nor any other checksum.
- o Extension headers are not widely supported in network hardware or devices. For instance, several NIC offloads don't work in the presence of extension headers.

5.2.4 Encapsulation techniques

Various encapsulation techniques have been proposed for implementing network virtualization and mobility. LISP is an example of an encapsulation that is based on locator identifier separation similar to ILA. The primary drawback of encapsulation is complexity and per packet overhead. For, instance when LISP is used with IPv6 the encapsulation overhead is fifty-six bytes and two IP headers are present in every packet. This adds considerable processing costs, requires considerations to handle path MTU correctly, and certain network accelerations may be lost.

6 IANA Considerations

There are no IANA considerations in this specification.

7 References

7.1 Normative References

- [RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.
- [RFC2460bis] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", draft-ietf-6man-rfc2460bis-03, January 2016.
- [RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006.
- [RFC6296] Wasserman, M. and F. Baker, "IPv6-to-IPv6 Network Prefix Translation", RFC 6296, June 2011.
- [RFC1071] Braden, R., Borman, D., Partridge, C., and W. Plummer, "Computing the Internet checksum", RFC 1071, September 1988.
- [RFC1624] Rijssinghani, A., "Computation of the Internet Checksum via Incremental Update", RFC 1624, May 1994.
- [RFC6724] Thaler, D., Ed., Draves, R., Matsumoto, A., and T. Chown, "Default Address Selection for Internet Protocol Version 6 (IPv6)", RFC 6724, September 2012.

7.2 Informative References

- [RFC6740] RJ Atkinson and SN Bhatti, "Identifier-Locator Network Protocol (ILNP) Architectural Description", RFC 6740, November 2012.
- [RFC6741] RJ Atkinson and SN Bhatti, "Identifier-Locator Network Protocol (ILNP) Engineering Considerations", RFC 6741, November 2012.
- [RFC1918] Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, February 1996.
- [RFC3363] Bush, R., Durand, A., Fink, B., Gudmundsson, O., and T. Hain, "Representing Internet Protocol version 6 (IPv6)

Addresses in the Domain Name System (DNS)", RFC 3363, August 2002.

- [RFC3587] Hinden, R., Deering, S., and E. Nordmark, "IPv6 Global Unicast Address Format", RFC 3587, August 2003.
- [RFC4193] Hinden, R. and B. Haberman, "Unique Local IPv6 Unicast Addresses", RFC 4193, October 2005.
- [RFC6144] Baker, F., Li, X., Bao, C., and K. Yin, "Framework for IPv4/IPv6 Translation", RFC 6144, April 2011.
- [NVO3ARCH] Black, D., Hudson, J., Kreeger, L., Lasserre, M., and Narten, T., "An Architecture for Overlay Networks (NVO3)", draft-ietf-nvo3-arch-03
- [GUE] Herbert, T., and Yong, L., "Generic UDP Encapsulation", draft-herbert-gue-02, work in progress.
- [GUESEC] Yong, L., and Herbert, T. "Generic UDP Encapsulation (GUE) for Secure Transport", draft-hy-gue-4-secure-transport-00, work in progress

8 Acknowledgments

The author would like to thank Mark Smith, Lucy Yong, Erik Kline, Saleem Bhatti, Petr Lapukhov, Blake Matheny, Doug Porter, Pierre Pfister, and Fred Baker for their insightful comments for this draft; Roy Bryant, Lorenzo Colitti, Mahesh Bandewar, and Erik Kline for their work on defining and applying ILA.

Appendix A: Communication scenarios

This section describes the use of identifier-locator addressing in several scenarios.

A.1 Terminology for scenario descriptions

A formal notation for identifier-locator addressing with ILNP is described in [RFC6740]. We extend this to include for network virtualization cases.

Basic terms are:

- A = IP Address
- I = Identifier
- L = Locator
- LUI = Locally unique identifier
- VNI = Virtual network identifier
- VA = An IPv4 or IPv6 virtual address
- VAX = An IPv6 networking identifier (IPv6 VA mapped to VAX)
- SIR = Prefix for standard identifier representation
- VNET = IPv6 prefix for a tenant (assumed to be globally routable)
- Iaddr = IPv6 address of an Internet host

An ILA IPv6 address is denoted by

L:I

A SIR address with a locally unique identifier and SIR prefix is denoted by

SIR:LUI

A virtual identifier with a virtual network identifier and a virtual IPv4 address is denoted by

VNI:VA

An ILA IPv6 address with a virtual networking identifier for IPv4 would then be denoted

L:(VNI:VA)

The local and remote address pair in a packet or endpoint is denoted

A,A

An address translation sequence from SIR addresses to ILA addresses

for transmission on the network and back to SIR addresses at a receiver has notation:

A,A -> L:I,A -> A,A

A.2 Identifier objects

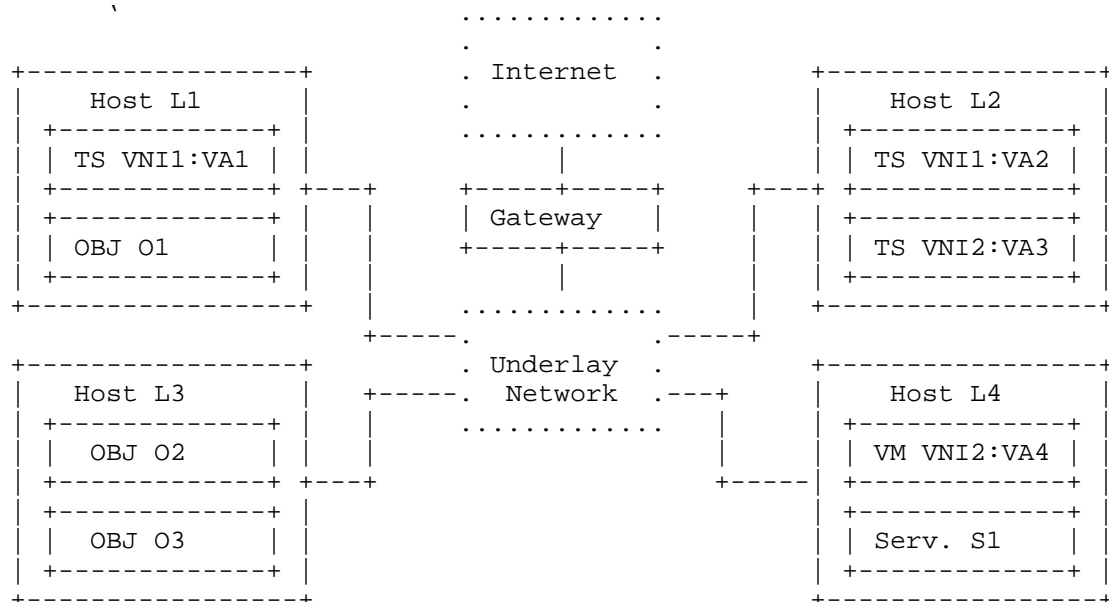
Identifier-locator addressing is broad enough in scope to address many different types of networking entities. For the purposes of this section we classify these as "objects" and "tenant systems".

Objects encompass uses where nodes are address by local unique identifiers (LUI). In the scenarios below objects are denoted by OBJ.

Tenant systems are those associated with network virtualization that have virtual addresses (that is they are addressed by VNI:VA). In the scenarios below tenant systems are denoted by TS.

A.3 Reference network for scenarios

The figure below provides an example network topology with ILA addressing in use. In this example, there are four hosts in the network with locators L1, L2, L3, and L4. There three objects with identifiers O1, O2, and O3, as well as a common networking service with identifier S1. There are two virtual networks VNI1 and VNI2, and four tenant systems addressed as: VA1 and VA2 in VNI1, VA3 and VA4 in VNI2. The network is connected to the Internet via a gateway.



Several communication scenarios can be considered:

- 1) Object to object
- 2) Object to Internet
- 3) Internet to object
- 4) Tenant system to local service
- 5) Object to tenant system
- 6) Tenant system to Internet
- 7) Internet to tenant system
- 8) IPv4 tenant system to service
- 9) Tenant system to tenant system same virtual network using IPv6
- 10) Tenant system to tenant system in same virtual network using IPv4
- 11) Tenant system to tenant system in different virtual network using IPv6
- 12) Tenant system to tenant system in different virtual network using IPv4
- 13) IPv4 tenant system to IPv6 tenant system in different virtual networks

A.4 Scenario 1: Object to task

The transport endpoints for object to object communication are the SIR addresses for the objects. When a packet is sent on the wire, the locator is set in the destination address of the packet. On reception the destination addresses is converted back to SIR representation for processing at the transport layer.

If object O1 is communicating with object O2, the ILA translation sequence would be:

```
SIR:O1,SIR:O2 ->           // Transport endpoints on O1
SIR:O1,L3:O2 ->            // ILA used on the wire
SIR:O1,SIR:O2              // Received at O2
```

A.5 Scenario 2: Object to Internet

Communication from an object to the Internet is accomplished through use of a SIR address (globally routable) in the source address of packets. No ILA translation is needed in this path.

If object O1 is sending to an address Iaddr on the Internet, the packet addresses would be:

```
SIR:O1,Iaddr
```

A.6 Scenario 3: Internet to object

An Internet host transmits a packet to a task using an externally routable SIR address. The SIR prefix routes the packet to a gateway for the datacenter. The gateway translates the destination to an ILA address.

If a host on the Internet with address Iaddr sends a packet to object O3, the ILA translation sequence would be:

```
Iaddr,SIR:O3 ->                // Transport endpoint at Iaddr
Iaddr,L1:O3 ->                  // On the wire in datacenter
Iaddr,SIR:O3                    // Received at O3
```

A.7 Scenario 4: Tenant system to service

A tenant can communicate with a datacenter service using the SIR address of the service.

If TS VA1 is communicating with service S1, the ILA translation sequence would be:

```
VNET:VA1,Saddr->                // Transport endpoints in TS
SIR:(VNET:VA1):Saddr->          // On the wire
SIR:(VNET:VA1):Saddr            // Received at S1
```

Where VNET is the address prefix for the tenant and Saddr is the IPv6 address of the service.

The ILA translation sequence in the reverse path, service to tenant system, would be:

```
Saddr,SIR:(VNET:VA1)            // Transport endpoints in S1
Saddr,L1:(VNET:VA1)            // On the wire
Saddr,VNET:VA1                  // Received at the TS
```

Note that from the point of view of the service task there is no material difference between a peer that is a tenant system versus one which is another task.

A.8 Scenario 5: Object to tenant system

An object can communicate with a tenant system through it's externally visible address.

If object O2 is communicating with TS VA4, the ILA translation sequence would be:

```
SIR:O2,VNET:VA4 ->                // Transport endpoints at T2
SIR:O2,L4:(VNI2:VAX4) ->          // On the wire
```

```
SIR:O2,VNET:VA4
```

```
// Received at TS
```

A.9 Scenario 6: Tenant system to Internet

Communication from a TS to the Internet assumes that the VNET for the TS is globally routable, hence no ILA translation would be needed.

If TS VA4 sends a packet to the Internet, the addresses would be:

```
VNET:VA4,Iaddr
```

A.10 Scenario 7: Internet to tenant system

An Internet host transmits a packet to a tenant system using an externally routable tenant prefix and address. The prefix routes the packet to a gateway for the datacenter. The gateway translates the destination to an ILA address.

If a host on the Internet with address Iaddr is sending to TS VA4, the ILA translation sequence would be:

```
Iaddr,VNET:VA4 ->
```

```
// Endpoint at Iaddr
```

```
Iaddr,L4:(VNI2:VAX4) ->
```

```
// On the wire in datacenter
```

```
Iaddr,VNET:VA4
```

```
// Received at TS
```

A.11 Scenario 8: IPv4 tenant system to object

A TS that is IPv4-only may communicate with an object using protocol translation. The object would be represented as an IPv4 address in the tenant's address space, and stateless NAT64 should be usable as described in [RFC6145].

If TS VA2 communicates with object O3, the ILA translation sequence would be:

```
VA2,ADDR3 ->
```

```
// IPv4 endpoints at TS
```

```
SIR:(VNI1:VA2),L3:O3 ->
```

```
// On the wire in datacenter
```

```
SIR:(VNI1:VA2),SIR:O3
```

```
// Received at task
```

VA2 is the IPv4 address in the tenant's virtual network, ADDR4 is an address in the tenant's address space that maps to the network service.

The reverse path, task sending to a TS with an IPv4 address, requires a similar protocol translation.

For object O3 communicate with TS VA2, the ILA translation sequence would be:

```
SIR:O3,SIR:(VNI1:VA2) ->           // Endpoints at T4
SIR:O3,L2:(VNI1:VA2) ->           // On the wire in datacenter
ADDR4,VA2                          // IPv4 endpoint at TS
```

A.12 Tenant to tenant system in the same virtual network

ILA may be used to allow tenants within a virtual network to communicate without the need for explicit encapsulation headers.

A.12.1 Scenario 9: TS to TS in the same VN using IPV6

If TS VA1 sends a packet to TS VA2, the ILA translation sequence would be:

```
VNET:VA1,VNET:VA2 ->               // Endpoints at VA1
VNET:VA1,L2:(VNI1,VAX2) ->         // On the wire
VNET:VA1,VNET:VA2 ->               // Received at VA2
```

A.12.2 Scenario 10: TS to TS in same VN using IPv4

For two tenant systems to communicate using IPv4 and ILA, IPv4/IPv6 protocol translation is done both on the transmit and receive.

If TS VA1 sends an IPv4 packet to TS VA2, the ILA translation sequence would be:

```
VA1,VA2 ->                         // Endpoints at VA1
SIR:(VNI1:VA1),L2:(VNI1,VA2) ->    // On the wire
VA1,VA2                             // Received at VA2
```

Note that the SIR is chosen by an ILA node as an appropriate SIR prefix in the underlay network. Tenant systems do not use SIR address for this communication, they only use virtual addresses.

A.13 Tenant system to tenant system in different virtual networks

A tenant system may be allowed to communicate with another tenant system in a different virtual network. This should only be allowed with explicit policy configuration.

A.13.1 Scenario 11: TS to TS in different VNs using IPV6

For TS VA4 to communicate with TS VA1 using IPv6 the translation sequence would be:

```
VNET2:VA4,VNET1:VA1->             // Endpoint at VA4
VNET2:VA4,L1:(VNI1,VAX1)->        // On the wire
VNET2:VA4,VNET1:VA1                // Received at VA1
```

Note that this assumes that VNET1 and VNET2 are globally routable between the two virtual networks.

A.13.2 Scenario 12: TS to TS in different VNs using IPv4

To allow IPv4 tenant systems in different virtual networks to communicate with each other, an address representing the peer would be mapped into each tenant's address space. IPv4/IPv6 protocol translation is done on transmit and receive.

For TS VA4 to communicate with TS VA1 using IPv4 the translation sequence may be:

```
VA4,SADDR1 ->                // IPv4 endpoint at VA4
SIR:(VNI2:VA4),L1:(VNI1,VA1)-> // On the wire
SADDR4,VA1                    // Received at VA1
```

SADDR1 is the mapped address for VA1 in VA4's address space, and SADDR4 is the mapped address for VA4 in VA1's address space.

A.13.3 Scenario 13: IPv4 TS to IPv6 TS in different VNs

Communication may also be mixed so that an IPv4 tenant system can communicate with an IPv6 tenant system in another virtual network. IPv4/IPv6 protocol translation is done on transmit.

For TS VA4 using IPv4 to communicate with TS VA1 using IPv6 the translation sequence may be:

```
VA4,SADDR1 ->                // IPv4 endpoint at VA4
SIR:(VNI2:VA4),L1:(VNI1,VAX1)-> // On the wire
SIR:(VNI2:VA4),VNET1:VA1        // Received at VA1
```

SADDR1 is the mapped IPv4 address for VA1 in VA4's address space.

In the reverse direction, TS VA1 using IPv6 would communicate with TS VA4 with the translation sequence:

```
VNET1:VA1,SIR:(VNI2:VA4)      // Endpoint at VA1
VNET1:VA1,L4:(VNI2:VA4)      // On the wire
SADDR1,VA4                    // Received at VA4
```

Appendix B: unique identifier generation

The unique identifier type of ILA identifiers can address 2^{60} objects. This appendix describes some method to perform allocation of identifiers for objects to avoid duplicated identifiers being allocated.

B.1 Globally unique identifiers method

For small to moderate sized deployments the technique for creating locally assigned global identifiers described in [RFC4193] could be used. In this technique a SHA-1 digest of the time of day in NTP format and an EUI-64 identifier of the local host is performed. N bits of the result are used as the globally unique identifier.

The probability that two or more of these IDs will collide can be approximated using the formula:

$$P = 1 - \exp(-N^2 / 2^{L+1})$$

where P is the probability of collision, N is the number of identifiers, and L is the length of an identifier.

The following table shows the probability of a collision for a range of identifiers using a 60-bit length.

Identifiers	Probability of Collision
1000	$4.3368 \cdot 10^{-13}$
10000	$4.3368 \cdot 10^{-11}$
100000	$4.3368 \cdot 10^{-09}$
1000000	$4.3368 \cdot 10^{-07}$

Note that locally unique identifiers may be ephemeral, for instance a task may only exist for a few seconds. This should be considered when determining the probability of identifier collision.

B.2 Universally Unique Identifiers method

For larger deployments, hierarchical allocation may be desired. The techniques in Universally Unique Identifier (UUID) URN ([RFC4122]) can be adapted for allocating unique object identifiers in sixty bits. An identifier is split into two components: a registrar prefix and sub-identifier. The registrar prefix defines an identifier block which is managed by an agent, the sub-identifier is a unique value within the registrar block.

For instance, each host in a network could be an agent so that unique identifiers for objects could be created autonomously by the host.

The identifier might be composed of a twenty-four bit host identifier followed by a thirty-six bit timestamp. Assuming that a host can allocate up to 100 identifiers per second, this allows about 21.8 years before wrap around.

```

/* LUI identifier with host registrar and timestamp */
|3 bits|1|      24 bits      |              36 bits              |
+-----+-----+-----+-----+
| 0x1  |C| Host identifier |              Timestamp Identifier      |
+-----+-----+-----+-----+

```

Appendix C: Datacenter task virtualization

This section describes some details to apply ILA to virtualizing tasks in a datacenter.

C.1 Address per task

Managing the port number space for services within a datacenter is a nontrivial problem. When a service task is created, it may run on arbitrary hosts. The typical scenario is that the task will be started on some machine and will be assigned a port number for its service. The port number must be chosen dynamically to not conflict with any other port numbers already assigned to tasks on the same machine (possibly even other instances of the same service). A canonical name for the service is entered into a database with the host address and assigned port. When a client wishes to connect to the service, it queries the database with the service name to get both the address of an instance as well as its port number. Note that DNS is not adequate for the service lookup since it does not provide port numbers.

With ILA, each service task can be assigned its own IPv6 address and therefore will logically be assigned the full port space for that address. This is a dramatic simplification since each service can now use a publicly known port number that does not need to be unique between services or instances. A client can perform a lookup on the service name to get an IP address of an instance and then connect to that address using a well known port number. In this case, DNS is sufficient for directing clients to instances of a service.

C.2 Job scheduling

In the usual datacenter model, jobs are scheduled to run as tasks on some number of machines. A distributed job scheduler provides the scheduling which may entail considerable complexity since jobs will often have a variety of resource constraints. The scheduler takes these constraints into account while trying to maximize utility of

the datacenter in terms utilization, cost, latency, etc. Datacenter jobs do not typically run in virtual machines (VMs), but may run within containers. Containers are mechanisms that provide resource isolation between tasks running on the same host OS. These resources can include CPU, disk, memory, and networking.

A fundamental problem arises in that once a task for a job is scheduled on a machine, it often needs to run to completion. If the scheduler needs to schedule a higher priority job or change resource allocations, there may be little recourse but to kill tasks and restart them on a different machine. In killing a task, progress is lost which results in increased latency and wasted CPU cycles. Some tasks may checkpoint progress to minimize the amount of progress lost, but this is not a very transparent or general solution.

An alternative approach is to allow transparent job migration. The scheduler may migrate running jobs from one machine to another.

C.3 Task migration

Under the orchestration of the job scheduler, the steps to migrate a job may be:

- 1) Stop running tasks for the job.
- 2) Package the runtime state of the job. The runtime state is derived from the containers for the jobs.
- 3) Send the runtime state of the job to the new machine where the job is to run.
- 4) Instantiate the job's state on the new machine.
- 5) Start the tasks for the job continuing from the point at which it was stopped.

This model similar to virtual machine (VM) migration except that the runtime state is typically much less data-- just task state as opposed to a full OS image. Task state may be compressed to reduce latency in migration.

C.3.1 Address migration

ILA facilitates address (specifically SIR address) migration between hosts as part of task migration or for other purposes. The steps in migrating an address might be:

- 1) Configure address on the target host.
- 2) Suspend use of the address on the old host. This includes handling established connections (see next section). A state may be established to drop packets or send ICMP destination

unreachable when packets to the migrated address are received.

- 3) Update the identifier to locator mapping database. Depending on the control plane implementation this may include pushing the new mapping to hosts.
- 4) Communicating hosts will learn of the new mapping via a control plane either by participation in a protocol for mapping propagation or by the ILA resolution protocol.

C.3.2 Connection migration

When a task and its addresses are migrated between machines, the disposition of existing TCP connections needs to be considered.

The simplest course of action is to drop TCP connections across a migration. Since migrations should be relatively rare events, it is conceivable that TCP connections could be automatically closed in the network stack during a migration event. If the applications running are known to handle this gracefully (i.e. reopen dropped connections) then this may be viable.

For seamless migration, open connections may be migrated between hosts. Migration of these entails pausing the connection, packaging connection state and sending to target, instantiating connection state in the peer stack, and restarting the connection. From the time the connection is paused to the time it is running again in the new stack, packets received for the connection should be silently dropped. For some period of time, the old stack will need to keep a record of the migrated connection. If it receives a packet, it should either silently drop the packet or forward it to the new location.

Author's Address

Tom Herbert
Facebook
1 Hacker Way
Menlo Park, CA
EMail: tom@herbertland.com

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: May 4, 2017

R. Winter
M. Faath
F. Weisshaar
University of Applied Sciences Augsburg
October 31, 2016

Privacy considerations for IP broadcast and multicast protocol designers
draft-ietf-intarea-broadcast-consider-01

Abstract

A number of application-layer protocols make use of IP broadcasts or multicast messages for functions like local service discovery or name resolution. Some of these functions can only be implemented efficiently using such mechanisms. When using broadcasts or multicast messages, a passive observer in the same broadcast/multicast domain can trivially record these messages and analyze their content. Therefore, broadcast/multicast protocol designers need to take special care when designing their protocols.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 4, 2017.

Copyright Notice

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

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

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

Table of Contents

1. Introduction	2
1.1. Requirements Language	3
2. Privacy considerations	3
2.1. Message frequency	4
2.2. Persistent identifiers	4
2.3. Anticipate user behavior	5
2.4. Consider potential correlation	5
2.5. Configurability	6
3. Operational considerations	7
4. Summary	7
5. Other considerations	8
6. Acknowledgments	8
7. IANA Considerations	8
8. Security Considerations	8
9. Informative References	9
Authors' Addresses	10

1. Introduction

Broadcast and multicast messages have a large (and to the sender unknown) receiver group by design. Because of that, these two mechanisms are vital for a number of basic network functions such as auto-configuration. Application developers use broadcast/multicast messages to implement things like local service or peer discovery and it appears that an increasing number of applications make use of it. And, as RFC 919 [RFC0919] puts it, "The use of broadcasts [...] is a good base for many applications".

Using broadcast/multicast can become problematic if the information that is being distributed can be regarded as sensitive or when the information that is distributed by multiple of these protocols can be correlated in a way that sensitive data can be derived. This is clearly true for any protocol, but broadcast/multicast is special in at least two respects:

- (a) The aforementioned large receiver group, consisting of receivers unknown to the sender. This makes eavesdropping without special privileges or a special location in the network trivial for anybody in the broadcast/multicast domain.

- (b) Encryption is more difficult when broadcast/multicast messages, leaving content of these messages in the clear and making it easier to spoof and replay them.

Given the above, privacy protection for protocols based on broadcast or multicast communication is significantly more difficult compared to unicast communication and at the same time invading the privacy is much easier.

Privacy considerations of IETF-specified protocols have received some attention in the recent past (e.g. RFC 7721 [RFC7721] or RFC 7919 [RFC7819]). There is also general guidance available for document authors on when and how to include a privacy considerations section in their documents and on how to evaluate the privacy implications of Internet protocols [RFC6973]. RFC6973 also describes potential threats to privacy in great detail and lists terminology that is also used in this document.

In contrast to RFC6973, this document contains a number of privacy considerations especially for broadcast/multicast protocol designers that are intended to reduce the likelihood that a broadcast/multicast protocol can be misused to collect sensitive data about devices, users and groups of users on a broadcast/multicast domain. These considerations particularly apply to protocols designed outside the IETF for two reasons. For one, non-standard protocols will likely not receive operational attention and support in making them more secure such as e.g. DHCP snooping does for DHCP because they typically are not documented. The other reason is that these protocols have been designed in isolation, where a set of considerations to follow is useful in the absence of a larger community providing feedback. In particular, carelessly designed broadcast/multicast protocols can break privacy efforts at different layers of the protocol stack such as MAC address or IP address randomization [RFC4941].

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Privacy considerations

There are a few obvious and a few not necessarily obvious things designers of broadcast/multicast protocols should consider in respect to the privacy implications of their protocol. Most of these items are based on protocol behavior observed as part of experiments on operational networks [TRAC2016].

2.1. Message frequency

Frequent broadcast/multicast traffic caused by an application can give user behavior and online times away. This allows a passive observer to potentially deduce a user's current activity (e.g. a game) and it allows to create an online profile (i.e. times the user is on the network). The higher the frequency of these messages, the more accurate this profile will be. Given that broadcasts/multicasts are only visible in the same broadcast/multicast domain, these messages also give the rough location of the user away (e.g. a campus or building).

This behavior has e.g. been observed by a synchronization mechanism of a popular application, where multiple messages have been sent per minute via broadcast. Given this behavior, it is possible to record a device's time on the network with a sub-minute accuracy given only the traffic of this single application installed on the device. But also services used for local name resolution in modern operating systems utilize broadcast/multicast protocols (e.g. mDNS, LLMNR or NetBIOS) to announce for example their shares regularly and allow a tracking of the online time of a device.

If a protocol relies on frequent or periodic broadcast/multicast messages, the frequency SHOULD be chosen conservatively, in particular if the messages contain persistent identifiers (see next subsection). Also, intelligent message suppression mechanisms such as the ones employed in mDNS [RFC6762] SHOULD be implemented. The lower the frequency of broadcast messages, the harder traffic analysis and surveillance becomes.

2.2. Persistent identifiers

A few broadcast/multicast protocols observed in the wild make use of persistent identifiers. This includes the use of host names or more abstract persistent identifiers such as a UUID or similar. These IDs, which e.g. identify the installation of a certain application might not change across updates of the software and are therefore extremely long lived. This allows a passive observer to track a user precisely if broadcast/multicast messages are frequent. This is even true in case the IP and/or MAC address changes. Such identifiers also allow two different interfaces (e.g. WiFi and Ethernet) to be correlated to the same device. If the application makes use of persistent identifiers for multiple installations of the same application for the same user, this even allows to infer that different devices belong to the same user.

The aforementioned broadcast messages from a synchronization mechanism of a popular application also included a persistent

identifier in every broadcast. This identifier did never change after the application was installed and allowed to track a device even when it changed its network interface or when it connected to a different network.

If a broadcast/multicast protocol relies on IDs to be transmitted, it SHOULD be considered if frequent ID rotations are possible in order to make user tracking more difficult. Persistent IDs are considered bad practice in general for broadcast and multicast communication as persistent application layer IDs will make efforts on lower layers to randomize identifiers (e.g. [I-D.huitema-6man-random-addresses]) useless or at least much more difficult.

2.3. Anticipate user behavior

A large number of users name their device after themselves, either using their first name, last name or both. Often a host name includes the type, model or maker of a device, its function or includes language specific information. Based on gathered data, this appears currently to be prevalent user behavior [TRAC2016]. For protocols using the host name as part of the messages, this clearly will reveal personally identifiable information to everyone on the local network. This information can also be used to mount more sophisticated attacks, when e.g. the owner of a device is identified (as an interesting target) or properties of the device are known (e.g. known vulnerabilities).

A popular operating system vendor includes the name the user chooses for the user account during the installation process as part of the host name of the device. The name of the operating system is also included, revealing therefore two pieces of information, which can be regarded as private information if the host name is used in broadcast/multicast messages.

Where possible, the use of host names and other user provided information in broadcast/multicast protocols SHOULD be avoided. If only a persistent ID is needed, this can be generated. An application might want to display the information it will broadcast on the LAN at install/config time, so the user is at least aware of the application's behavior. More host name considerations can be found in [I-D.ietf-intarea-hostname-practice]. More information on user participation can be found in RFC 6973 [RFC6973].

2.4. Consider potential correlation

A large number of services and applications make use of the broadcast/multicast mechanism. That means there are various sources of information that are easily accessible by a passive observer. In

isolation, the information these protocols reveal might seem harmless, but given multiple such protocols, it might be possible to correlate this information. E.g. a protocol that uses frequent messages including a UUID to identify the particular installation does not give the identity of the user away. But a single message including the user's host name might just do that and it can be correlated using e.g. the MAC address of the device's interface.

In the experiments described in [TRAC2016], it was possible to correlate frequently sent broadcast messages that included a unique identifier with other broadcast/multicast messages containing usernames (e.g. mDNS, LLMNR or NetBIOS), but also relationships to other users. This allowed to reveal the real identity of the users of many devices but it also gave some information about their social environment away.

A broadcast protocol designer should be aware of the fact that even if - in isolation - the information a protocol leaks seems harmless, there might be ways to correlate that information with other broadcast protocol information to reveal sensitive information about a user.

2.5. Configurability

A lot of applications and services using broadcast/multicast protocols do not include the means to declare "safe" environments (e.g. based on the SSID of a WiFi network and the MAC addresses of the access points). E.g. a device connected to a public WiFi will likely broadcast the same information as when connected to the home network. It would be beneficial if certain behavior could be restricted to "safe" environments.

A popular operating system e.g. allows the user to specify the trust level of the network the device connects to, which for example restricts specific system services (using broadcast/multicast messages for their normal operation) to be used in untrusted networks. Such functionality could implemented as part of an application.

An application developer making use of broadcasts/multicasts as part of the application SHOULD make the broadcast feature, if possible, configurable, so that potentially sensitive information does not leak on public networks, where the thread to privacy is much larger.

3. Operational considerations

Besides changing end-user behavior, choosing sensible defaults as an operating system vendor (e.g. for suggesting host names) and the considerations for protocol designers mentioned in this document, there are things that the network administrators/operators can do to limit the above mentioned problems.

A feature not uncommonly found on access points e.g. is to filter broadcast and multicast traffic. This will potentially break certain applications or some of their functionality but will also protect the users from potentially leaking sensitive information.

4. Summary

Increasingly, applications rely on broadcast and multicast messages. For some, broadcasts/multicasts are the basis of their application logic, others use broadcasts/multicasts to improve certain aspects of the application but are fully functional in case broadcasts/multicasts fail. Irrespective of the role of broadcast and multicast messages for the application, the designers of protocols that make use of them should be very careful in their protocol design because of the special nature of broad- and multicast.

It is not always possible to implement certain functionality via unicast, but in case a protocol designer chooses to rely on broadcast/multicast, the following should be carefully considered:

- o IETF-specified protocols, such as mDNS [RFC6762], should be used if possible as operational support might exist to protect against the leakage of private information
- o Avoid using user-specified information inside broadcast/multicast messages as users will often use personal information or other information aiding attackers, in particular if the user is unaware about how that information is being used
- o Avoid persistent IDs in messages as this allows user tracking, correlation and potentially has a devastating effect on other privacy protection mechanisms
- o If you really must use a broadcast/multicast protocol and cannot use an IETF-specified protocol, then:
 - * Be very conservative in how frequently you send messages as an effort in data minimization

- * Seek advice from IETF-specifies protocols such as message suppression in mDNS
- * Try to design the protocol in a way that the information cannot be correlated with other information in broadcast/multicast messages
- * Let the user configure safe environments if possible (e.g. based on the SSID)

[Note: discussions on this document should be take place on the Intarea mailing list of the IETF. Subscription:
<https://www.ietf.org/mailman/listinfo/int-area>, Mailing list archive:
<https://www.ietf.org/mail-archive/web/int-area/current/maillist.html>]

5. Other considerations

Besides the privacy implications of frequent broadcasting, it also represents a performance problem. In particular in certain wireless technologies such as 802.11, broadcast and multicast are transmitted at a much lower rate (the lowest common denominator rate) compared to unicast and therefore have a much bigger impact on the overall available airtime. Further, it will limit the ability for devices to go to sleep if frequent broadcasts are being sent. A similar problem in respect to Router Advertisements is addressed in [I-D.ietf-v6ops-reducing-ra-energy-consumption]. In that respect broadcasts can be used for another class of attacks that not related to privacy. The potential impact on network performance should nevertheless be considered by broadcast protocol designers.

6. Acknowledgments

We would like to thank Eliot Lear and Stephane Bortzmeyer for their input.

This work was partly supported by the European Commission under grant agreement FP7-318627 mPlane. Support does not imply endorsement.

7. IANA Considerations

This memo includes no request to IANA.

8. Security Considerations

This document deals with privacy-related considerations of broadcast- and multicast-based protocols. It contains advice for designers of such protocols to minimize the leakage of privacy-sensitive information. The intent of the advice is to make sure that

identities will remain anonymous and user tracking will be made difficult.

9. Informative References

- [I-D.huitema-6man-random-addresses]
Huitema, C., "Implications of Randomized Link Layers Addresses for IPv6 Address Assignment", draft-huitema-6man-random-addresses-03 (work in progress), March 2016.
- [I-D.ietf-intarea-hostname-practice]
Huitema, C. and D. Thaler, "Current Hostname Practice Considered Harmful", draft-ietf-intarea-hostname-practice-00 (work in progress), October 2015.
- [I-D.ietf-v6ops-reducing-ra-energy-consumption]
Yourtchenko, A. and L. Colitti, "Reducing energy consumption of Router Advertisements", draft-ietf-v6ops-reducing-ra-energy-consumption-03 (work in progress), November 2015.
- [RFC0919] Mogul, J., "Broadcasting Internet Datagrams", STD 5, RFC 919, DOI 10.17487/RFC0919, October 1984, <<http://www.rfc-editor.org/info/rfc919>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC4941] Narten, T., Draves, R., and S. Krishnan, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 4941, DOI 10.17487/RFC4941, September 2007, <<http://www.rfc-editor.org/info/rfc4941>>.
- [RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, DOI 10.17487/RFC6762, February 2013, <<http://www.rfc-editor.org/info/rfc6762>>.
- [RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <<http://www.rfc-editor.org/info/rfc6973>>.
- [RFC7721] Cooper, A., Gont, F., and D. Thaler, "Security and Privacy Considerations for IPv6 Address Generation Mechanisms", RFC 7721, DOI 10.17487/RFC7721, March 2016, <<http://www.rfc-editor.org/info/rfc7721>>.

[RFC7819] Jiang, S., Krishnan, S., and T. Mrugalski, "Privacy Considerations for DHCP", RFC 7819, DOI 10.17487/RFC7819, April 2016, <<http://www.rfc-editor.org/info/rfc7819>>.

[TRAC2016] Faath, M., Weisshaar, F., and R. Winter, "How Broadcast Data Reveals Your Identity and Social Graph", 7th International Workshop on TRaffic Analysis and Characterization IEEE TRAC 2016, September 2016.

Authors' Addresses

Rolf Winter
University of Applied Sciences Augsburg
Augsburg
DE

Email: rolf.winter@hs-augsburg.de

Michael Faath
University of Applied Sciences Augsburg
Augsburg
DE

Email: michael.faath@hs-augsburg.de

Fabian Weisshaar
University of Applied Sciences Augsburg
Augsburg
DE

Email: fabian.weisshaar@hs-augsburg.de

Internet Area WG
Internet-Draft
Intended status: Standard track
Expires May 4, 2017

T. Herbert
Facebook
L. Yong
Huawei USA
O. Zia
Microsoft
October 31, 2016

Generic UDP Encapsulation
draft-ietf-intarea-gue-00

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), 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/ietf/lid-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>

This Internet-Draft will expire on May 4, 2017.

Copyright Notice

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

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

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 Generic UDP Encapsulation (GUE), which is a scheme for using UDP to encapsulate packets of different 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 data fields as part of the encapsulation, and is generic in that it can encapsulate packets of various IP protocols.

Table of Contents

1. Introduction	5
1.1 Terminology	5
2. Base packet format	7
2.1. GUE version	7
3. Version 0	7
3.1. Header format	8
3.2. Proto/ctype field	9
3.2.1 Proto field	9
3.2.2 Ctype field	10
3.3. Flags and extension fields	10
3.3.1. Requirements	10
3.3.2. Example GUE header with extension fields	11
3.4. Private data	12
3.5. Message types	12
3.5.1. Control messages	12
3.5.2. Data messages	13
3.6. Hiding the transport layer protocol number	13
4. Version 1	14
4.1. Direct encapsulation of IPv4	14
4.2. Direct encapsulation of IPv6	15
5. Operation	15
5.1. Network tunnel encapsulation	16
5.2. Transport layer encapsulation	16
5.3. Encapsulator operation	16
5.4. Decapsulator operation	17
5.4.1. Processing a received data message	17
5.4.2. Processing a received control message	18
5.5. Router and switch operation	18
5.6. Middlebox interactions	18
5.6.1. Connection semantics	19
5.6.2. NAT	19
5.7. Checksum Handling	19
5.7.1. Requirements	19

5.7.2. UDP Checksum with IPv4	20
5.7.3. UDP Checksum with IPv6	20
5.8. MTU and fragmentation	21
5.9. Congestion control	21
5.10. Multicast	21
5.11. Flow entropy for ECMP	22
5.11.1. Flow classification	22
5.11.2. Flow entropy properties	23
5.12. Negotiation of acceptable flags and extension fields	24
6. Motivation for GUE	24
6.1. Benefits of GUE	24
6.2. Comparison of GUE to other encapsulations	25
7. Security Considerations	26
8. IANA Consideration	27
8.1. UDP source port	27
8.2. GUE version number	27
8.3. Control types	27
8.4. Flag-fields	28
9. Acknowledgements	29
10. References	29
10.1. Normative References	29
10.2. Informative References	30
Appendix A: NIC processing for GUE	32
A.1. Receive multi-queue	32
A.2. Checksum offload	33
A.2.1. Transmit checksum offload	33
A.2.2. Receive checksum offload	34
A.3. Transmit Segmentation Offload	34
A.4. Large Receive Offload	35
Appendix B: Implementation considerations	36
B.1. Privileged ports	36
B.2. Setting flow entropy as a route selector	36
B.3. Hardware protocol implementation considerations	36
Authors' Addresses	37

1. Introduction

This specification describes Generic UDP Encapsulation (GUE) which is 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.

GUE provides an extensible header format for including optional data in the encapsulation header. This 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 data in the encapsulation header. This feature can be used by a site or implementation to define local custom optional data, and allows experimentation of options that may eventually become standard.

This document does not define any specific GUE extensions. [GUEEXTENS] specifies a set of core extensions and [GUE4NVO3] defines an extension for using GUE with network virtualization.

The motivation for the GUE protocol is described in section 6.

1.1 Terminology

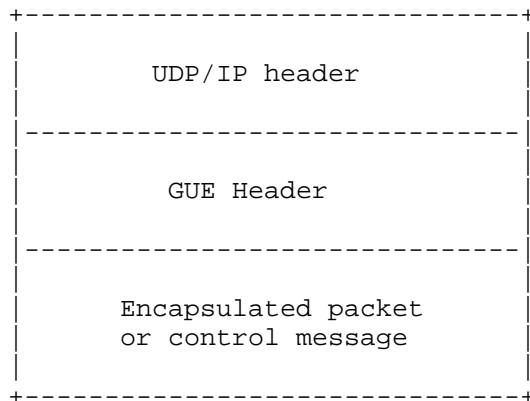
GUE	Generic UDP Encapsulation
GUE Header	A variable length protocol header that is composed of a primary four byte header and zero or more four byte words for optional header data
GUE packet	A UDP/IP packet that contains a GUE header and GUE payload within the UDP payload
Encapsulator	A network node that encapsulates a packet in GUE
Decapsulator	A network node that decapsulates and processes packets encapsulated in GUE
Data message	An encapsulated packet in the GUE payload that is addressed to the protocol stack for an associated protocol
Control message	A formatted message in the GUE payload that is

implicitly addressed to a decapsulator to monitor or control the state or behavior of a tunnel

Flags	A set of bit flags in the primary GUE header
Extension field	An optional field in a GUE header whose presence is indicated by corresponding flag(s)
C-bit	A single bit flag in the primary GUE header that indicates whether the GUE packet contains a control message or not.
Hlen	A field in the primary GUE header that gives the length of the GUE header
Proto/ctype	A field in the GUE header that holds either the IP protocol number for a data message or a type for a control message
Private data	Optional data in the GUE header that may be used for private purposes
Outer IP header	Refers to the outer most IP header of a packet when encapsulating a packet over IP
Inner IP header	Refers to an encapsulated IP header when an IP packets is encapsulated
Outer packet	Refers to an encapsulating packet
Inner packet	Refers to a packet that is encapsulated
Tunnel	An abstraction of a path across a network that ships packets or protocols across a network that normally wouldn't support them. Tunnels provide communication paths between two endpoints. Encapsulation is one common technique used to actualize tunnels
Overlay network	A computer network that is built on top of another network
Underlay network	A network over which an overlay network is built

2. Base packet format

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:



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

2.1. GUE version

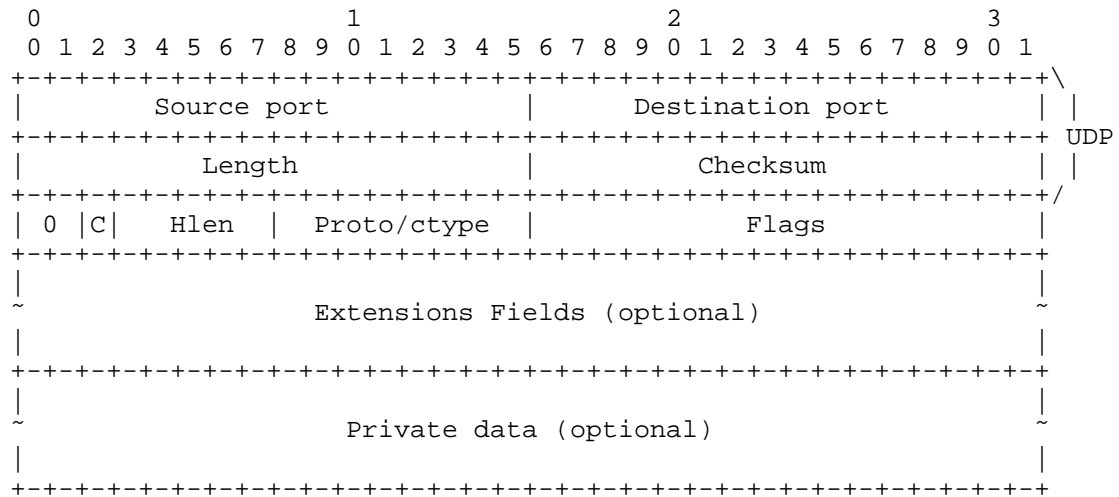
The first two bits of the GUE header contain the GUE protocol version number. The rest of the fields after the GUE version number are defined based on the version number. Versions 0 and 1 are described in this specification; versions 2 and 3 are reserved.

3. Version 0

Version 0 of GUE defines a generic extensible format to encapsulate packets by Internet protocol number.

3.1. Header format

The header format for version 0 of GUE in UDP is:



The contents of the UDP header are:

- o Source port: If connection semantics (section 5.6.1) are applied to an encapsulation, this is set to the source port in the local tuple. When connection semantics are not applied this should be set to a flow entropy value for use with ECMP; the properties of flow entropy are described in section 5.11.
- o Destination port: If connection semantics (section 5.6.1) are applied to an encapsulation, this is set to the destination port for the tuple. If connection semantics are not applied this is set to the GUE assigned port number, 6080.
- o Length: Canonical length of the UDP packet (length of UDP header and payload).
- o Checksum: Standard UDP checksum (handling is described in section 5.7).

The GUE header consists of:

- o Ver: GUE protocol version (0).
- o C: C-bit. When set indicates a control message, not set indicates a data message.

- o Hlen: Length in 32-bit words of the GUE header, including optional extension fields but not the first four bytes of the header. Computed as $(\text{header_len} - 4) / 4$. All GUE headers are a multiple of four bytes in length. Maximum header length is 128 bytes.
- o Proto/ctype: When the C-bit is set this field contains a control message type for the payload (section 3.2.2). When C-bit is not set, the field holds the Internet protocol number for the encapsulated packet in the payload (section 3.2.1). The control message or encapsulated packet begins at the offset provided by Hlen.
- o Flags. Header flags that may be allocated for various purposes and may indicate presence of extension fields. Undefined header flag bits MUST be set to zero on transmission.
- o Extension Fields: Optional fields whose presence is indicated by corresponding flags.
- o Private data: Optional private data (see section 3.4). If private data is present it immediately follows that last extension field present in the header. The length of this data is determined by subtracting the starting offset from the header length.

3.2. Proto/ctype field

The proto/ctype field contains the type of the GUE payload. This can either be an IP protocol number or a control message type number. Intermediate devices may parse the GUE payload per the number in the proto/ctype field, and header flags cannot affect the interpretation of the proto/ctype field.

3.2.1 Proto field

When the C-bit is not set the proto/ctype field contains an IANA Internet Protocol Number. The protocol number is interpreted relative to the IP protocol that encapsulates the UDP packet (i.e. protocol of the outer IP header).

When the outer IP protocol is IPv4 the proto field may be set to any number except for those that refer to IPv6 extension headers or ICMPv6 options (number 58). An exception is that the destination options extension header using the PadN option may be used with IPv4 as described in section 3.6. The "no next header" protocol number (59) may be used with IPv4 as described below.

When the outer IP protocol is IPv6 the proto field may be set to any defined protocol number except Hop-by-hop options (number 0). If a received GUE packet in IPv6 contains a protocol number that is an extension header (e.g. Destination Options) then the extension header is processed after the GUE header as though the GUE header itself were an extension header.

IP protocol number 59 ("No next header") may be set to indicate that the GUE payload does not begin with the header of an IP protocol. This would be the case, for instance, if the GUE payload were a fragment when performing GUE level fragmentation. The interpretation of the payload is performed through other means (such as flags and extension fields), and intermediate devices must not parse packets based on the IP protocol number in this case.

3.2.2 Ctype field

When the C-bit is set, the proto/ctype field must be set to a valid control message type. A value of zero indicates that the GUE payload requires further interpretation to deduce the control type. This might be the case when the payload is a fragment of a control message, where only the reassembled packet can be interpreted as a control message.

Control message types 1 through 127 may be defined in standards. Types 128 through 255 are reserved to be user defined for experimentation or private control messages.

This document does not specify any standard control message types other than type 0.

3.3. Flags and extension fields

Flags and associated extension fields are the primary mechanism of extensibility in GUE. As mentioned in section 3.1 GUE header flags may indicate the presence of optional extension fields in the GUE header. [GUEXTENS] defines a basic set of GUE extensions.

3.3.1. Requirements

There are sixteen flag bits in the GUE header. A flag may indicate presence of an extension fields. The size of an extension field indicated by a flag must be fixed.

Flags may be paired together to allow different lengths for an extension 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

3.4. Private data

An implementation may use private data for its own use. The private data immediately follows the last extension field in the GUE header and is not a fixed length. This data is considered part of the GUE header and must be accounted for in header length (Hlen). The length of the private data must be a multiple of four and is determined by subtracting the offset of private data in the GUE header from the header length. Specifically:

$$\text{Private_length} = (\text{Hlen} * 4) - \text{Length}(\text{flags})$$

Where "Length(flags)" returns the sum of lengths of all the extension fields present in the GUE header. When there is no private data present, the length of the private data is zero.

The semantics and interpretation of private data are implementation specific. The private data may be structured as necessary, for instance it might contain its own set of flags and extension fields.

An encapsulator and decapsulator MUST agree on the meaning of private data before using it. The mechanism to achieve this agreement is outside the scope of this document but could include implementation-defined behavior, coordinated configuration, in-band communication using GUE control messages, or out-of-band messages.

If a decapsulator receives a GUE packet with private data, it MUST validate the private data appropriately. If a decapsulator does not expect private data from an encapsulator the packet MUST be dropped. If a decapsulator cannot validate the contents of private data per the provided semantics the packet MUST also be dropped. An implementation may place security data in GUE private data which must be verified for packet acceptance.

3.5. Message types

3.5.1. Control messages

Control messages carry formatted message that are implicitly addressed to the decapsulator to monitor or control the state or behavior of a tunnel (OAM). For instance, an echo request and corresponding echo reply message may be defined to test for liveness.

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 control messages can be created that follow the same path as data messages.

3.5.2. Data messages

Data messages carry encapsulated packets that are addressed to the protocol stack for the associated protocol. Data messages are a primary means of encapsulation and can be used to create tunnels for overlay networks.

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

3.6. Hiding the transport layer protocol number

The GUE header indicates the Internet protocol of the encapsulated packet. This is either contained in the Proto/ctype field of the primary GUE header, or is contained in the Payload Type field of a GUE Transform Field (used to encrypt the payload with DTLS, [GUESEC]). If the protocol number must be obfuscated, that is the transport protocol in use must be hidden from the network, then a trivial destination options can be used at the beginning of the payload.

The PadN destination option can be used to encode the transport protocol as a next header of an extension header (and maintain alignment of encapsulated transport headers). The Proto/ctype field or Payload Type field of the GUE Transform field is set to 60 to indicate that the first encapsulated header is a Destination Options extension header.

The format of the extension header is below:

```

+-----+
| Next Header | 2 | 1 | 0 |
+-----+
```

For IPv4, it is permitted in GUE to use this precise destination option to contain the obfuscated protocol number. In this case next header must refer to a valid IP protocol for IPv4. No other extension headers or destination options are permitted with IPv4.

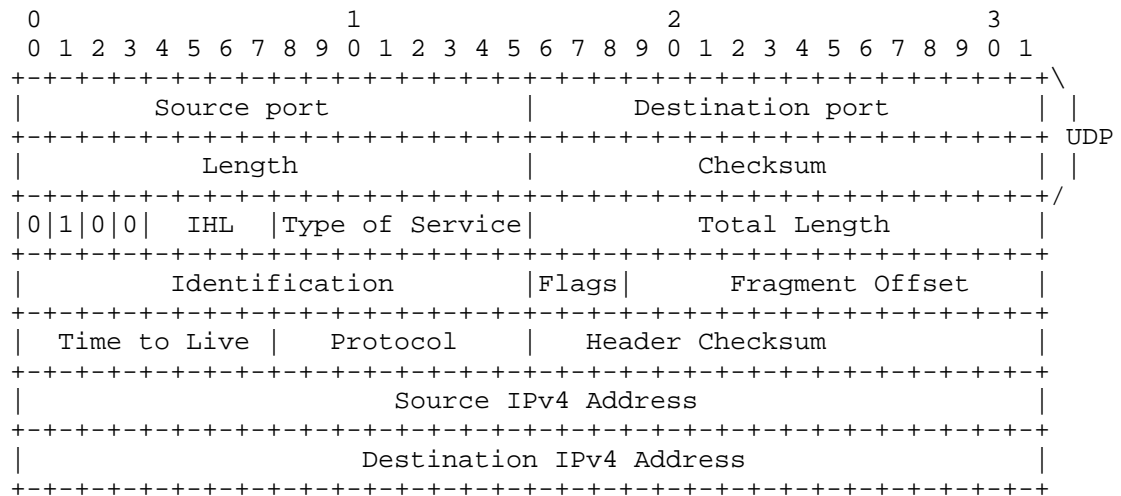
4. Version 1

Version 1 of GUE allows direct encapsulation of IPv4 and IPv6 in UDP. In this version there is no GUE header; a UDP packet encapsulates an IP packet. The first two bits of the UDP payload for GUE are the GUE version and coincide with the first two bits of the version number in the IP header. The first two version bits of IPv4 and IPv6 are 01, so we use GUE version 1 for direct IP encapsulation which makes two bits of GUE version to also be 01.

This technique is effectively a means to compress out the GUE header when encapsulating IPv4 or IPv6 packets and there are no flags or extension fields present. This method is compatible to use on the same port number as packets with the GUE header (GUE version 0 packets). This technique saves encapsulation overhead on costly links for the common use of IP encapsulation, and also obviates the need to allocate a separate port number for IP-over-UDP encapsulation.

4.1. Direct encapsulation of IPv4

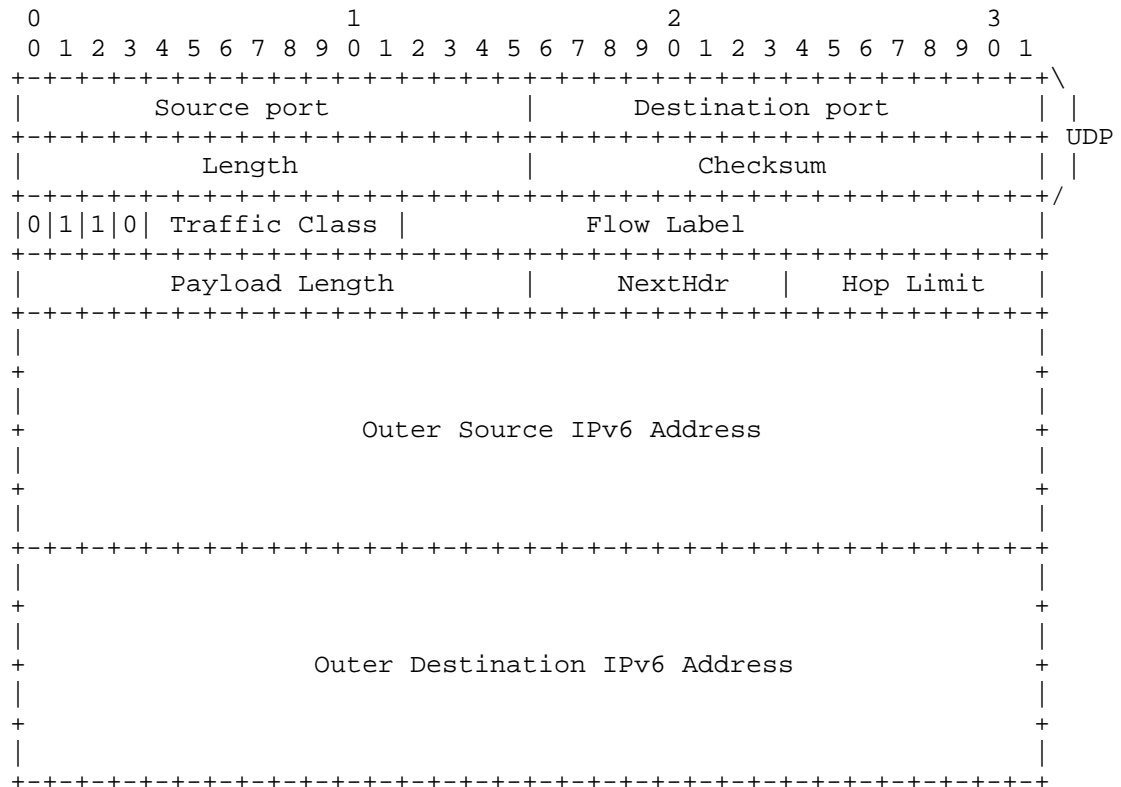
The format for encapsulating IPv4 directly in UDP is demonstrated below:



Note that 0100 value IP version field expresses the GUE version as 1 (bits 01) and IP version as 4 (bits 0100).

4.2. Direct encapsulation of IPv6

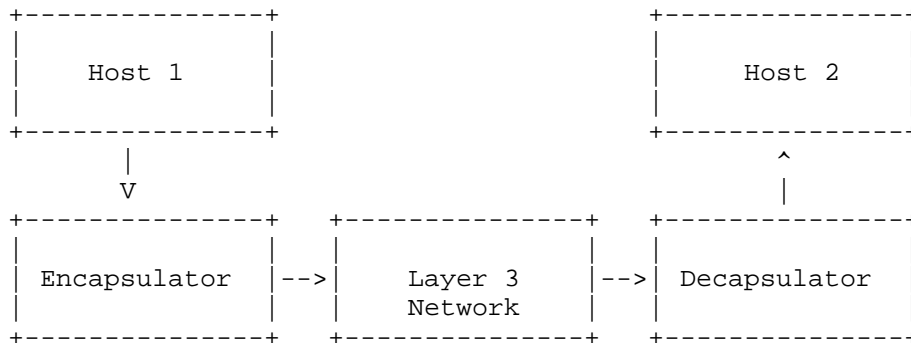
The format for encapsulating IPv4 directly in UDP is demonstrated below:



Note that 0110 value IP version field expresses the GUE version as 1 (bits 01) and IP version as 6 (bits 0110).

5. Operation

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



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

5.1. 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 hosts.

5.2. Transport layer encapsulation

When encapsulating layer 4 packets, the encapsulator and decapsulator should be co-resident with the hosts. 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. Note that the transport layer ports in the encapsulated packet are independent of the UDP ports in the outer packet.

Details about performing transport layer encapsulation are discussed in [TOU].

5.3. Encapsulator operation

Encapsulators create GUE data messages, set the fields of the UDP header, set flags and optional extension 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 hosts (routers implementing tunnels for instance). In either case, the intended target (decapsulator) is indicated by the outer destination IP address and destination port in the UDP header.

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 protocol over another. For instance, if an IP packet is being encapsulated in GUE then diffserv interaction [RFC2983] and ECN propagation for tunnels [RFC6040] should be followed.

5.4. Decapsulator operation

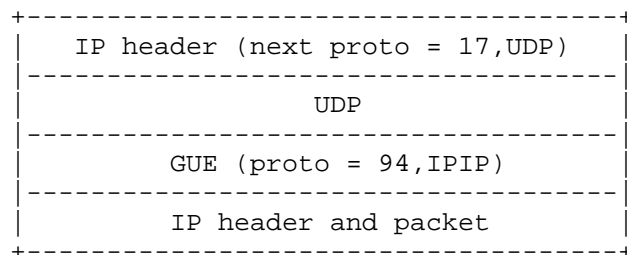
A decapsulator performs decapsulation of GUE packets. A decapsulator is addressed by the outer destination IP address of a GUE packet. The decapsulator validates packets, including fields of the GUE header.

If a decapsulator receives a GUE packet with an unsupported version, unknown flag, bad header length (too small for included extension fields), unknown control message type, bad protocol number, an unsupported Proto/ctype, or an otherwise malformed header, it MUST drop the packet. Such events may be logged subject to configuration and rate limiting of logging messages. No error message is returned back to the encapsulator. Note that set flags in GUE that are unknown to a decapsulator MUST NOT be ignored. If a GUE packet is received by a decapsulator with unknown flags, the packet MUST be dropped.

5.4.1. Processing a received data message

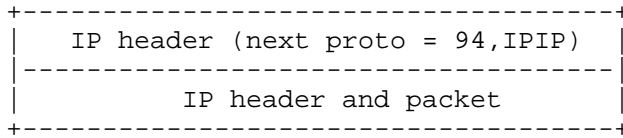
If a valid data message is received the UDP and GUE headers are removed from the packet. The outer IP header remains in tact and the next protocol in the header is set to the protocol from the proto field in the GUE header. The resulting packet is then resubmitted into the protocol stack to process that packet as though it was received with the protocol in the GUE header.

As an example, consider that a data message is received where GUE encapsulates an IP packet. In this case proto field in the GUE header is set 94 for IPIP:



The receiver removes the UDP and GUE headers and sets the next

protocol field in the IP packet to IPIP which is derived from the GUE proto field. The resultant packet would have the format:



This packet is then resubmitted into the protocol stack to be processed as an IPIP packet.

5.4.2. Processing a received control message

If a valid control message is received the packet must be processed as a control message. The specific processing to be performed depends on the ctype in the GUE header.

5.5. 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 normally need to parse a GUE header, and none of the flags or extension fields in the GUE header should affect routing.

An intermediate node SHOULD NOT modify a GUE header or GUE payload when forwarding packets since correctly identifying GUE packets in the network based on port numbers is not robust (see [RFC7605]). An intermediate node may encapsulate a GUE packet in another GUE packet, for instance to implement a network tunnel (i.e. by encapsulating an IP packet with a GUE payload in another IP packet as a GUE payload). In this case the router takes the role of an encapsulator, and the corresponding decapsulator is the logical endpoint of the tunnel. When encapsulating a GUE packet within another GUE packet, there are no provisions to automatically copy flags or extension fields to the outer GUE header. Each layer of encapsulation is considered independent.

5.6. Middlebox interactions

A middle box may interpret some flags and extension fields of the GUE header for classification purposes, but is not required to understand any of the flags or extension fields in GUE packets. A middle box must 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 extension

fields.

5.6.1. Connection semantics

A middlebox may infer bidirectional connection semantics for 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. In this case the UDP source port for a tunnel would be a fixed value for a tunnel and not set to be flow entropy as described in section 5.11.

The selection of whether to make the UDP source port fixed or set to a flow entropy value for each packet sent should be configurable for a tunnel.

5.6.2. 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 in section 5.6.1. GUE endpoints may also invoke STUN [RFC5389] or ICE [RFC5245] to manage NAT port mappings for encapsulations.

5.7. Checksum Handling

The potential for mis-delivery of packets due to corruption of IP, UDP, or GUE headers must be considered. Historically, the UDP checksum would be considered sufficient as a check against corruption of either the UDP header and payload or the IP addresses. Encapsulation protocols, such as GUE, may be originated or terminated on devices incapable of computing the UDP checksum for packet. This section discusses the requirements around checksum and alternatives that might be used when an endpoint does not support UDP checksum.

5.7.1. Requirements

One of the following requirements must be met:

- o UDP checksums are enabled (for IPv4 or IPv6).

- o The GUE header checksum is used (defined in [GUEEXTENS]).
- o Use zero UDP checksums. This is always permissible with IPv4, in IPv6 they may only be used in accordance with applicable requirements in [GREUDP], [RFC6935], and [RFC6936].

5.7.2. UDP Checksum with IPv4

For UDP in IPv4, the UDP checksum MUST be processed as specified in [RFC768] and [RFC1122] for both transmit and receive. An encapsulator MAY set the UDP checksum to zero for performance or implementation considerations. The IPv4 header includes a checksum that protects against mis-delivery of the packet due to corruption of IP addresses. The UDP checksum potentially provides protection against corruption of the UDP header, GUE header, and GUE payload. Enabling or disabling the use of checksums is a deployment consideration that should take into account the risk and effects of packet corruption, and whether the packets in the network are already adequately protected by other, possibly stronger mechanisms such as the Ethernet CRC. If an encapsulator sets a zero UDP checksum for IPv4 it SHOULD use the GUE header checksum as described in [GUEEXTENS].

When a decapsulator receives a packet, the UDP checksum field MUST be processed. If the UDP checksum is non-zero, the decapsulator MUST verify the checksum before accepting the packet. By default a decapsulator SHOULD accept UDP packets with a zero checksum. A node MAY be configured to disallow zero checksums per [RFC1122]; this may be done selectively, for instance disallowing zero checksums from certain hosts that are known to be sending over paths subject to packet corruption. If verification of a non-zero checksum fails, a decapsulator lacks the capability to verify a non-zero checksum, or a packet with a zero-checksum was received and the decapsulator is configured to disallow, the packet MUST be dropped.

5.7.3. UDP Checksum with IPv6

In IPv6 there is no checksum in the IPv6 header that protects against mis-delivery due to address corruption. Therefore, when GUE is used over IPv6, either the UDP checksum must be enabled, the GUE header checksum must be used, or a zero UDP checksum is used if applicable requirements are met. Setting a zero checksum may be desirable for performance or implementation reasons, in which case the GUE header checksum MUST be used or requirements for using zero UDP checksums in [RFC6935] and [RFC6936] MUST be met. If the UDP checksum is enabled, then the GUE header checksum should not be used since it is mostly redundant.

When a decapsulator receives a packet, the UDP checksum field MUST be

processed. If the UDP checksum is non-zero, the decapsulator MUST verify the checksum before accepting the packet. By default a decapsulator MUST only accept UDP packets with a zero checksum if the GUE header checksum is used and is verified. If verification of a non-zero checksum fails, a decapsulator lacks the capability to verify a non-zero checksum, or a packet with a zero-checksum and no GUE header checksum was received, the packet MUST be dropped.

5.8. MTU and fragmentation

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 UDP source port. An operator should set MTU to account for encapsulation overhead and reduce the likelihood of fragmentation.

Alternative to IP fragmentation, the GUE fragmentation extension can be used. GUE fragmentation is described in [GUEEXTENS].

5.9. 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 whether a transmitter will consistently implement proper congestion control, then congestion control at the encapsulation layer MUST be provided per RFC5405. 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 [CIRCBRK], or traffic isolation may be used to provide rudimentary congestion control. For finer grained congestion control that allows alternate congestion control algorithms, reaction time within an RTT, and interaction with ECN, in-band mechanisms may be warranted.

5.10. Multicast

GUE packets may be multicast to decapsulators using a multicast destination address in the encapsulating IP headers. Each receiving

host will decapsulate the packet independently following normal decapsulator operations. The receiving decapsulators should agree on the same set of GUE parameters and properties; how such an agreement is reached is outside the scope of this document.

GUE allows encapsulation of unicast, broadcast, or multicast traffic. Flow entropy (the value in the UDP source port) may be generated from the header of encapsulated unicast or broadcast/multicast packets at an encapsulator. The mapping mechanism between the encapsulated multicast traffic and the multicast capability in the IP network is transparent and independent of the encapsulation and is otherwise outside the scope of this document.

5.11. Flow entropy for ECMP

5.11.1. 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.

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 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 and connection semantics are not applied, the source port in the outer UDP packet is set to a flow entropy value that corresponds to 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.

Examples of deriving flow entropy for encapsulation are:

- o If the encapsulated packet is a layer 4 packet, TCP/IPv4 for instance, the flow entropy 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 flow entropy 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 flow entropy 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.

[RFC6438] discusses methods to compute and flow entropy value for IPv6 flow labels, those methods can also be used to create flow entropy values for GUE.

5.11.2. Flow entropy properties

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

- o The value set in the source port should be within the ephemeral port range (49152 to 65535 [RFC6335]). Since the high order two bits of the port are set to one this provides fourteen bits of entropy for the value.
- o The flow entropy should have a uniform distribution across encapsulated flows.
- o An encapsulator may occasionally change the flow entropy used for an inner flow per its discretion (for security, route selection, etc). To avoid thrashing or flapping the value, the flow entropy used for a flow should not change more than once every thirty seconds (or a configurable value).
- o Decapsulators, or any networking devices, should not attempt to interpret flow entropy as anything more than an opaque value. Neither should they attempt to reproduce the hash calculation used by an encapsulator in creating a flow entropy value. 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 flow entropy calculation 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.
- o The assignment function for flow entropy should be randomly seeded to mitigate denial of service attacks. The seed may be changed periodically.

5.12. Negotiation of acceptable flags and extension fields

An encapsulator and decapsulator must achieve agreement about GUE parameters that will be used in communications. Parameters include GUE versions, flags and optional extension fields that can be used, security algorithms and keys, supported protocols and control messages, etc. This document proposes different general methods to accomplish this, the details of implementing these are considered out of scope.

General methods for this are:

- o Configuration. The parameters used for a tunnel are configured at each endpoint.
- o Negotiation. A tunnel negotiation can be performed. This could be accomplished in-band of GUE using control messages or private data.
- o Via a control plane. Parameters for communicating with a tunnel endpoint can be set in a control plane protocol (such as that needed for nvo3).
- o Via security negotiation. If security is used that would typically imply a key exchange between endpoints. Other GUE parameters may be conveyed as part of that process.

6. Motivation for GUE

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

6.1. Benefits of GUE

- * GUE is a generic encapsulation protocol. GUE can encapsulate protocols that are represented by an IP protocol number. This includes layer 2, layer 3, and layer 4 protocols.
- * GUE is an extensible encapsulation protocol. Standardized optional data such as security, virtual networking identifiers, fragmentation are being defined.
- * GUE allows private data to be sent as part of the encapsulation. This permits experimentation or customization in deployment.
- * GUE allows sending of control messages such as OAM using the same GUE header format (for routing purposes) as normal data messages.

- * GUE maximizes deliverability of non-UDP and non-TCP protocols.
- * GUE provides a means for exposing per flow entropy for ECMP for atypical protocols such as SCTP, DCCP, ESP, etc.

6.2. Comparison of GUE to other encapsulations

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 [RFC7637] and VXLAN [RFC7348] 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 [RFC7348], LISP [RFC6830] which encapsulates layer 3 packets, MPLS/UDP [7510], 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 for flow entropy as input to ECMP are defined in section 5.11.
- o GUE permits encapsulation of arbitrary IP protocols, which includes layer 2 3, and 4 protocols.
- 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 various IP protocols in UDP with minimal overhead (four bytes of additional header).
- o GUE is extensible. New flags and extension 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 data in the encapsulation header allows 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).
- o The flags-field model facilitates efficient implementation of extensibility in hardware.

For instance a TCAM can be use to parse a known set of N flags where the number of entries in the TCAM is 2^N .

By comparison, the number of TCAM entries needed to parse a set of N arbitrarily ordered TLVS is:

$$N! + (N-1)(N-1)! + (N-2)(N-2)! + \dots + (2)2! + (1)1!$$

7. Security Considerations

There are two important considerations of security with respect to GUE.

- o Authentication and integrity of the GUE header
- o Authentication, integrity, and confidentiality of the GUE payload.

Security is integrated into GUE by the use of GUE security related extensions; these are defined in [GUEEXTENS]. These extensions include methods to authenticate the GUE header and encrypt the GUE payload.

IPsec in transport mode may be used to authenticate or encrypt GUE packets (GUE header and payload). Existing network security mechanisms, such as address spoofing detection, DDOS mitigation, and transparent encrypted tunnels can be applied to GUE packets.

A hash function for computing flow entropy (section 5.11) should be randomly seeded to mitigate some possible denial service attacks.

8. IANA Consideration

8.1. UDP source port

A user UDP port number assignment for GUE has been assigned:

```
Service Name: gue
Transport Protocol(s): UDP
Assignee: Tom Herbert <therbert@google.com>
Contact: Tom Herbert <therbert@google.com>
Description: Generic UDP Encapsulation
Reference: draft-herbert-gue
Port Number: 6080
Service Code: N/A
Known Unauthorized Uses: N/A
Assignment Notes: N/A
```

8.2. GUE version number

IANA is requested to set up a registry for the GUE version number. The GUE version number is 2 bits containing four possible values. This document defines version 0 and 1. New values are assigned via Standards Action [RFC5226].

Version number	Description	Reference
0	Version 0	This document
1	Version 1	This document
2..3	Unassigned	

8.3. Control types

IANA is requested to set up a registry for the GUE control types. Control types are 8 bit values. New values for control types 1-127 are assigned via Standards Action [RFC5226].

Control type	Description	Reference
0	Need further interpretation	This document
1..127	Unassigned	
128..255	User defined	This document

8.4. Flag-fields

IANA is requested to create a "GUE flag-fields" registry to allocate flags and extension fields used with GUE. This shall be a registry of bit assignments for flags, length of extension fields for corresponding flags, and descriptive strings. There are sixteen bits for primary GUE header flags (bit number 0-15). New values are assigned via Standards Action [RFC5226].

Flags bits	Field size	Description	Reference
Bit 0	4 bytes	VNID	[GUE4NVO3]
Bit 1..3	001->8 bytes 010->16 bytes 011->32 bytes	Security	[GUEEXTENS]
Bit 4	8 bytes	Fragmen- tation	[GUEEXTENS]
Bit 5	4 bytes	Payload transform	[GUEEXTENS]
Bit 6	4 bytes	Remote checksum offload	[GUEEXTENS]
Bit 7	4 bytes	Checksum	[GUEEXTENS]
Bit 8..15		Unassigned	

New flags are to be allocated from high to low order bit contiguously without holes.

9. Acknowledgements

The authors would like to thank David Liu, Erik Nordmark, Fred Templin, Adrian Farrel, and Bob Briscoe for valuable input on this draft.

10. References

10.1. Normative References

- [RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <<http://www.rfc-editor.org/info/rfc768>>.
- [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <<http://www.rfc-editor.org/info/rfc1122>>.
- [RFC2434] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 2434, DOI 10.17487/RFC2434, October 1998, <<http://www.rfc-editor.org/info/rfc2434>>.
- [RFC2983] Black, D., "Differentiated Services and Tunnels", RFC 2983, DOI 10.17487/RFC2983, October 2000, <<http://www.rfc-editor.org/info/rfc2983>>.
- [RFC6040] Briscoe, B., "Tunnelling of Explicit Congestion Notification", RFC 6040, DOI 10.17487/RFC6040, November 2010, <<http://www.rfc-editor.org/info/rfc6040>>.
- [RFC6935] Eubanks, M., Chimento, P., and M. Westerlund, "IPv6 and UDP Checksums for Tunneled Packets", RFC 6935, DOI 10.17487/RFC6935, 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, DOI 10.17487/RFC6936, April 2013, <<http://www.rfc-editor.org/info/rfc6936>>.
- [RFC4459] Savola, P., "MTU and Fragmentation Issues with In-the-Network Tunneling", RFC 4459, DOI 10.17487/RFC4459, April 2006, <<http://www.rfc-editor.org/info/rfc4459>>.

10.2. Informative References

- [RFC3828] Larzon, L-A., Degermark, M., Pink, S., Jonsson, L-E., Ed., and G. Fairhurst, Ed., "The Lightweight User Datagram Protocol (UDP-Lite)", RFC 3828, July 2004, <<http://www.rfc-editor.org/info/rfc3828>>.
- [RFC7348] Mahalingam, M., Dutt, D., Duda, K., Agarwal, P., Kreeger, L., Sridhar, T., Bursell, M., and C. Wright, "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks", RFC 7348, August 2014, <<http://www.rfc-editor.org/info/rfc7348>>.
- [RFC7605] Touch, J., "Recommendations on Using Assigned Transport Port Numbers", BCP 165, RFC 7605, DOI 10.17487/RFC7605, August 2015, <<http://www.rfc-editor.org/info/rfc7605>>.
- [RFC7637] Garg, P., Ed., and Y. Wang, Ed., "NVGRE: Network Virtualization Using Generic Routing Encapsulation", RFC 7637, DOI 10.17487/RFC7637, September 2015, <<http://www.rfc-editor.org/info/rfc7637>>.
- [RFC7510] Xu, X., Sheth, N., Yong, L., Callon, R., and D. Black, "Encapsulating MPLS in UDP", RFC 7510, DOI 10.17487/RFC7510, April 2015, <<http://www.rfc-editor.org/info/rfc7510>>.
- [RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, DOI 10.17487/RFC4340, March 2006, <<http://www.rfc-editor.org/info/rfc4340>>.
- [RFC4787] Audet, F., Ed., and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, DOI 10.17487/RFC4787, January 2007, <<http://www.rfc-editor.org/info/rfc4787>>.
- [RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, DOI 10.17487/RFC5389, October 2008, <<http://www.rfc-editor.org/info/rfc5389>>.
- [RFC5285] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, DOI 10.17487/RFC5245, April 2010, <<http://www.rfc-editor.org/info/rfc5245>>.

- [RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, DOI 10.17487/RFC5405, November 2008, <<http://www.rfc-editor.org/info/rfc5405>>.
- [RFC7605] Touch, J., "Recommendations on Using Assigned Transport Port Numbers", BCP 165, RFC 7605, DOI 10.17487/RFC7605, August 2015, <<http://www.rfc-editor.org/info/rfc7605>>.
- [RFC6438] Carpenter, B. and S. Amante, "Using the IPv6 Flow Label for Equal Cost Multipath Routing and Link Aggregation in Tunnels", RFC 6438, DOI 10.17487/RFC6438, November 2011, <<http://www.rfc-editor.org/info/rfc6438>>.
- [RFC2003] Perkins, C., "IP Encapsulation within IP", RFC 2003, DOI 10.17487/RFC2003, October 1996, <<http://www.rfc-editor.org/info/rfc2003>>.
- [RFC3948] Huttunen, A., Swander, B., Volpe, V., DiBurro, L., and M. Stenberg, "UDP Encapsulation of IPsec ESP Packets", RFC 3948, DOI 10.17487/RFC3948, January 2005, <<http://www.rfc-editor.org/info/rfc3948>>.
- [RFC6830] Farinacci, D., Fuller, V., Meyer, D., and D. Lewis, "The Locator/ID Separation Protocol (LISP)", RFC 6830, DOI 10.17487/RFC6830, January 2013, <<http://www.rfc-editor.org/info/rfc6830>>.
- [RFC3378] Housley, R. and S. Hollenbeck, "EtherIP: Tunneling Ethernet Frames in IP Datagrams", RFC 3378, DOI 10.17487/RFC3378, September 2002, <<http://www.rfc-editor.org/info/rfc3378>>.
- [RFC2784] Farinacci, D., Li, T., Hanks, S., Meyer, D., and P. Traina, "Generic Routing Encapsulation (GRE)", RFC 2784, DOI 10.17487/RFC2784, March 2000, <<http://www.rfc-editor.org/info/rfc2784>>.
- [RFC4023] Worster, T., Rekhter, Y., and E. Rosen, Ed., "Encapsulating MPLS in IP or Generic Routing Encapsulation (GRE)", RFC 4023, DOI 10.17487/RFC4023, March 2005, <<http://www.rfc-editor.org/info/rfc4023>>.
- [RFC2661] Townsley, W., Valencia, A., Rubens, A., Pall, G., Zorn, G., and B. Palter, "Layer Two Tunneling Protocol "L2TP"", RFC 2661, DOI 10.17487/RFC2661, August 1999, <<http://www.rfc-editor.org/info/rfc2661>>.

- [GUEEXTENS] Herbert, T., Yong, L., and Templin, F., "Extensions for Generic UDP Encapsulation" draft-herbert-gue-extensions-00
- [GUE4NVO3] Yong, L., Herbert, T., Zia, O., "Generic UDP Encapsulation (GUE) for Network Virtualization Overlay" draft-hy-nvo3-gue-4-nvo-03
- [GUESEC] Yong, L., Herbert, T., "Generic UDP Encapsulation (GUE) for Secure Transport" draft-hy-gue-4-secure-transport-03
- [TCPUDP] Chesire, S., Graessley, J., and McGuire, R., "Encapsulation of TCP and other Transport Protocols over UDP" draft-cheshire-tcp-over-udp-00
- [TOU] Herbert, T., "Transport layer protocols over UDP" draft-herbert-transports-over-udp-00
- [GREUDP] Crabbe, E., Yong, L., Xu, X., and Herbert, T., "Generic UDP Encapsulation for IP Tunneling" draft-ietf-tsvwg-gre-in-udp-encap-19
- [GUT] Manner, J., Varia, N., and Briscoe, B., "Generic UDP Tunnelling (GUT) draft-manner-tsvwg-gut-02.txt"
- [CIRCBRK] Fairhurst, G., "Network Transport Circuit Breakers", draft-ietf-tsvwg-circuit-breaker-15
- [LCO] Cree, E., <https://www.kernel.org/doc/Documentation/networking/checksum-offloads.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 UDP based encapsulation.

This appendix is informational and does not constitute a normative part of this document.

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 flow entropy in the UDP 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, GUE fragmentation, 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.

If an encapsulator is co-resident with a host, then checksum offload

may be performed using remote checksum offload (described in [GUEEXTENS]). 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.

Another alternative when an encapsulator is co-resident with a host is to perform Local Checksum Offload [LCO]. In this method the inner transport layer checksum is offloaded and the outer UDP checksum can be deduced based on the fact that the portion of the packet covered by the inner transport checksum will sum to zero (or at least the bit wise not of the inner pseudo header).

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 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 extension fields should not contain values that must be updated on a per segment basis-- for example, extension 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: Implementation considerations

This appendix is informational and does not constitute a normative part of this document.

B.1. Privileged ports

Using the source port to contain a flow entropy 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.

B.2. Setting flow entropy as a route selector

An encapsulator generating flow entropy in the UDP source port 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 flow entropy. 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 flow entropy for a flow should be subject to hysteresis (at most once every thirty seconds) to limit the number of out of order packets.

B.3. 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
Facebook
1 Hacker Way
Menlo Park, CA 94052
US

Email: tom@herbertland.com

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

Email: lucy.yong@huawei.com

Osama Zia
Microsoft
1 Microsoft Way
Redmond, WA 98029
US

Email: osamaz@microsoft.com

INTAREA
Internet-Draft
Intended status: Standards Track
Expires: April 24, 2017

S. Kanugovi
S. Vasudevan
Nokia
F. Baboescu
Broadcom
J. Zhu
Intel
October 21, 2016

Multiple Access Management Protocol
draft-kanugovi-intarea-mams-protocol-01

Abstract

A communication network includes an access network element that delivers data to/from the user and an associated core network element providing connectivity with the application servers. Multiconnectivity scenarios are common where a device can be simultaneously connected to multiple communication networks based on different technology implementations and network architectures like WiFi, LTE, DSL. A smart combination and selection of access and core network paths that can dynamically adapt to changing network conditions can improve quality of experience for a user in a multiconnectivity scenario and improve overall network utilization and efficiency. This document presents the problem statement and proposes solution principles. It specifies the requirements and reference architecture for a multi access management services framework that can be used to flexibly select the best combination of uplink and downlink access and core network paths, ensuring better network efficiency and enhanced application performance.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 24, 2017.

Copyright Notice

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

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

Table of Contents

1. Conventions used in this document	3
2. Introduction	3
3. Terminology	3
4. Problem Statement	4
5. Solution Principles	5
6. Requirements	5
6.1. Access technology agnostic interworking	6
6.2. Support common transport deployments	6
6.3. Independent Access path selection for Uplink and Downlink	6
6.4. IP anchor selection independent of uplink and downlink access	6
6.5. Adaptive network path selection	6
6.6. Multipath support and Aggregation of access link capacities	6
6.7. Scalable mechanism based on IP interworking	7
6.8. Separate Control and Data plane functions	7
6.9. Lossless Path (Connection) Switching	7
6.10. Concatenation and Fragmentation to adapt to MTU differences	7
6.11. Configuring network middleboxes based on negotiated protocols	8
6.12. Client policy	8
6.13. Control plane signaling	8
6.14. Service discovery and reachability	8
7. MAMS Reference Architecture	8
8. MAMS call flow	11
9. Implementation considerations	12
10. Applicability to Mobile Edge Computing	12
11. Security Considerations	13
11.1. Data and Control plane security	13
12. Contributors	13

13. References	13
13.1. Normative References	13
13.2. Informative References	14
Authors' Addresses	14

1. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Introduction

Multi Access Management Signaling (MAMS) is a programmable framework that allows to select and configure network paths, as well as adapt to dynamic network conditions, when multiple network connections can serve a client device. It is based on principles of user plane interworking that enables the solution to be deployed as an overlay without impacting the underlying networks. The framework provides for mechanisms for flexible selection of network paths based on application needs that can be leverage network intelligence and policies to dynamically adapt to changing conditions.

The document presents the requirements, solution principles and functional architecture for the MAMS framework. MAMS mechanisms are not dependent on any specific network and transport protocols like TCP, UDP, GRE, MPTCP etc. It co-exists and complements the existing protocols by providing a way to negotiate and configure the protocols based on client and network capabilities. Further it allows exchange of network state information and leveraging network intelligence to optimize the performance of such protocols.

An important goal for MAMS is to ensure that there is minimal or no dependency on the actual access technology of the participating links, beyond the fact that there is IP connectivity between the access nodes. This allows the scheme to be scalable for addition of newer accesses and for independent technology evolution of the existing accesses.

3. Terminology

"Client": The end-user device supporting connections with multiple access nodes, possibly over different access technologies.

"Multiconnectivity Client": A client with multiple network connections.

"Access network functional element": The functional element in the network that delivers user data packets to the client via an access link like WiFi airlink, LTE airlink, DSL.

"Core": The functional element that anchors the client's IP address used for communication with applications via the network.

"User Plane Gateway": The functional element that can intercept and route user data packets.

"Network Connection manager"(NCM): A functional entity in the network that oversees distribution of data packets over the multiple available access and core network paths.

"Client Connection Manager" (CCM): A functional entity in the client that exchanges MAMS Signaling with the Network Connection Manager and configures the multiple network paths for transport of user data.

"Network Multi Access Data Proxy" (N-MADP): This functional entity in the network handles the user data traffic forwarding across multiple network paths. N-MADP is responsible for MAMS-specific u-plane functionalities in the network.

"Client Multi Access Data Proxy" (C-MADP): This functional entity in the client handles the user data traffic forwarding across multiple network paths. C-MADP is responsible for MAMS-specific u-plane functionalities in the client.

4. Problem Statement

Typically, a device has access to multiple communication networks based on different technologies, say LTE, WiFi, DSL, MuLTEfire, for accessing application services. Different technologies exhibit benefits and limitations in different scenarios. For example, WiFi leverages the large spectrum available in unlicensed spectrum to deliver high capacities at low cost in uncongested scenarios with small user population, but can show significant degradation in application performance in congested scenarios with large user population. Another example is LTE network, the capacity of which is often constrained by high cost and limited availability of the licensed spectrum, but offers predictable service even in multi-user scenarios due to controlled scheduling and licensed spectrum usage.

Additionally, the use of a particular access network path is often coupled with the use of the associated core network. For example, in an enterprise that has deployed WiFi and LTE communications network, enterprise applications, like printers, Corporate Audio and Video conferencing, are accessible only via WiFi access connected to the

enterprise hosted WiFi core, whereas the LTE access can be used to get LTE operator core anchored services including access to public internet.

Application performance in different scenarios, therefore becomes dependent on the choice of the communication networks due to the tight coupling of the access and the core network paths. Therefore to leverage the best possible application performance in the widest possible scenarios, a framework is needed that allows flexible selection of the combination of access and core network paths for application data delivery.

For example, in uncongested scenarios, it would be beneficial to use WiFi access in the uplink and downlink for connecting to enterprise applications. Whereas in congested scenarios, where use of WiFi in uplink by multiple users can lead to degraded capacity and increased delays due to contention, it would be beneficial to use scheduled LTE uplink access combined with WiFi downlink.

5. Solution Principles

This document proposes a Multiple Access Management Services(MAMS) framework for dynamic selection of uplink and downlink access and core network paths for a device connected to multiple communication networks. The multiple communication networks interwork at the user plane. The selection of paths is based on negotiation of capabilities and network link quality between the device and a functional element in the network, namely the network connection manager (NCM). NCM has the intelligence to setup and offer the best network path based on device and network capabilities, application needs and knowledge of the network state.

The NCM communicates with the Client Connection Manager (CCM), a functional element in the device, for negotiation, sharing information on the network path conditions, and configuring usage of the network paths. The messages between the NCM and CCM are carried as user plane data over any of the available network paths between the NCM and CCM.

6. Requirements

The requirements set out in this section are for the behavior of the MAMS mechanism and the related functional elements.

6.1. Access technology agnostic interworking

The access nodes can be of different technology types like LTE, WiFi etc. Since MAMS routes user plane data packets at the IP layer, which makes it agnostic to the type of underlying technology used at the access nodes.

6.2. Support common transport deployments

The network path selection and user plane distribution should work transparently across transport deployments that include e2e IPsec, VPNs, and middleboxes like NATs and proxies.

6.3. Independent Access path selection for Uplink and Downlink

IP layer routing enables the client to transmit on uplink using one access and receive data on downlink using another access, allowing client and network connection manager to select the access paths for uplink and downlink independent of each other.

6.4. IP anchor selection independent of uplink and downlink access

A client is able to flexibly negotiate the IP anchor, core network, independent of the access paths used to reach the IP anchor depending on the application needs.

6.5. Adaptive network path selection

The NCM node has the ability to determine the quality of each of the network paths, e.g. access link delay and capacity. The network path quality information is fed into the logic for selection of combination of network paths to be used for transporting user data. The path selection algorithm can use network path quality information, in addition to other considerations like network policies, for optimizing network usage and enhancing QoE delivered to the user.

6.6. Multipath support and Aggregation of access link capacities

MAMS supports distribution and aggregation of user data across multiple network paths at the IP layer. MAMS allows the client to leverage the combined capacity of the multiple network connections by enabling simultaneous transport of user data over multiple network paths. If required, packet re-ordering is done at the receiver, client (C-MADP) and/or the network (N-MADP), when user data packets are received out of order. MAMS allows flexibility to choose the flow steering and aggregation protocol based on capabilities supported by the client and the network data plane entities, C-MADP

and N-MADP respectively. A MAMS multi-connection aggregation solution should support existing transport and network layer protocols like TCP, UDP, GRE. If flow aggregation functions are realized using existing protocols such as Multi-Path TCP(MPTCP) and SCTP, MAMS framework should allow use and configuration of these aggregation protocols.

6.7. Scalable mechanism based on IP interworking

The mechanism is based on IP interworking, requiring only the IP connectivity between the access nodes and the interworking functionality is based on generically available IP routing and tunneling capabilities. This makes solution easy to deploy and scale easily when different networks are added and removed.

6.8. Separate Control and Data plane functions

The client negotiates with a network connection manager the choice of access for both uplink and downlink accesses and the IP anchor(core). The network connection manager configures the actual user data distribution function residing in the Anchor element, thus maintaining a clear separation between the control and data plane functions. This makes the MAMS framework amenable to SDN based architecture and implementations.

6.9. Lossless Path (Connection) Switching

When switching data traffic from one path (connection) to another, packets may be lost or delivered out-of-order, which will have negative impacts on the performance of higher layer protocols, e.g. TCP. MAMS should provide necessary mechanisms to ensure in-order delivery at the receiver, as well as support retransmissions at the transmitter during path switching.

6.10. Concatenation and Fragmentation to adapt to MTU differences

MAMS should support heterogeneous access networks, which may have different MTU sizes. Moreover, tunneling protocols also have a big impact on the MTU size. Hence, MAMS should support concatenation such that multiple IP packets may be encapsulated into a single packet to improve efficiency. MAMS should also support fragmentation such that a single IP packet may be fragmented and encapsulated into multiple ones to avoid IP fragmentation.

6.11. Configuring network middleboxes based on negotiated protocols

MAMS enables identification of the optimal parameters that may be used for configuring the middle-boxes, like binding expiry times and supported MTUs, for efficient operation of the user plane protocols, depending on the data plane related parameters negotiated between the client and the network. e.g. Configuring longer binding expiry time in NATs when UDP transport is used in contrast to the scenario where TCP is configured at the transport layer.

6.12. Client policy

MAMS framework should support consideration of policies at the client, in addition to guidance from the network in determination of network paths selected for different application services.

6.13. Control plane signaling

MAMS control signaling is carried over the user plane and is transparent to the transport protocols. MAMS should support delivery of control signaling over the existing Internet protocols, e.g. TCP or UDP.

6.14. Service discovery and reachability

MAMS offers the flexibility for the functional entity NCM to be collocated with any of the network elements and reachable via any of the available user plane paths. MAMS framework allows the flexibility for the CCM to choose one of the available NCM's and exchange control plane signaling over any of the available user plane paths. The choice of NCM can be based on considerations like, but not limited to, quality of link through with the NCM is reachable, Client preference or pre-configuration etc.

7. MAMS Reference Architecture

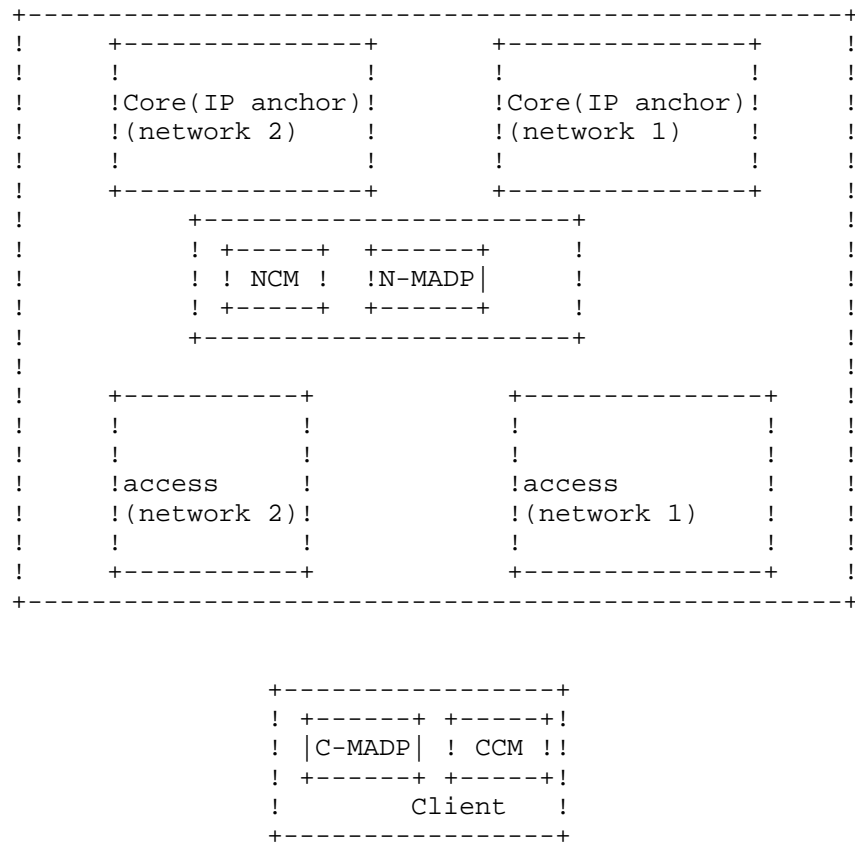


Figure 1: MAMS Reference Architecture

Figure 1 illustrates MAMS architecture for the scenario of a client served by 2 networks. The NCM and MADP, functional elements, are introduced for supporting MAMS mechanisms. The architecture is extendable to combine more than 2 networks, as well as any choice of participating network types (e.g. LTE, WLAN, MuLTEfire, DSL) and deployment architectures (e.g. with user plane gateway function at the access edge).

The N-MADP entity, at the network, handles the user data traffic forwarding across multiple network paths, as well as other user-plane functionalities, e.g. encapsulation, fragmentation, concatenation, reordering, retransmission, etc. N-MADP is the distribution node for uplink and downlink data with visibility of packets at the IP layer. Identification and distribution rules for different user data traffic type at the N-MADP are configured by the NCM. The NCM configures the routing in the N-MADP based on signaling exchanged with the CCM in

the client. In the UL, NCM specifies the core network path to be used by MADP to route uplink user data at the N-MADP. In the DL, NCM specifies the access links to be used for delivery of data to the client at the N-MADP.

The scheduling and load balancing algorithm at the N-MADP is configured by the NCM, based on static and/or dynamic network policies like assigning access and core paths for specific user data traffic type, data volume based percentage distribution, and link availability and feedback information from exchange of MAMS signaling with the CCM at the Client.

At the client, the Client Connection Manager (CCM) manages the multiple network connections. CCM is responsible for exchange of MAMS signaling messages with the NCM for supporting functions like configuring UL and DL user network path configuration for transporting user data packets, link sounding and reporting to support adaptive network path selection by NCM. In the downlink, for the user data received by the client, it configures IP layer forwarding for application data packet received over any of the accesses to reach the appropriate application module on the client. In the uplink, for the data transmitted by the client, it configures the routing table to determine the best access to be used for uplink data based on a combination of local policy and network policy delivered by the NCM. The C-MADP entity handles all MAMS-specific user-plane functionalities at the client, e.g. encapsulation, fragmentation, concatenation, reordering, retransmissions, etc. C-MADP is configured by CCM based on signaling exchange with NCM and local policies at the client.

A user plane tunnel, e.g. IPsec, may be needed for transporting user data packets between the N-MADP at the network and the C-MADP at the client. The user plane tunnel is needed to ensure security and routability of the user plane packets between the N-MADP and the C-MADP. The most common implementation of the user plane tunnel is the IPsec. C-MADP receives the configuration from CCM indicates, to C-MADP, the access network interfaces over which the IPsec tunnel needs to be established, and for each of the indicated interfaces, the parameters (e.g. N-MADP IPsec endpoint IP address reachable via the indicated access network interface) for setting up the IPsec tunnel. C-MADP sets up the IPsec tunnel with the N-MADP via each of the indicated access network interfaces, using appropriate signaling, say IKEv2 and parameters provided by the CCM. In deployments where the access node belonging to the two networks are connected via a secure and direct IP path, user plane tunnel may not be needed. Notice that the method for transporting user data packets between the N-MADP and the C-MADP should be general and based on the existing protocols.

8. MAMS call flow

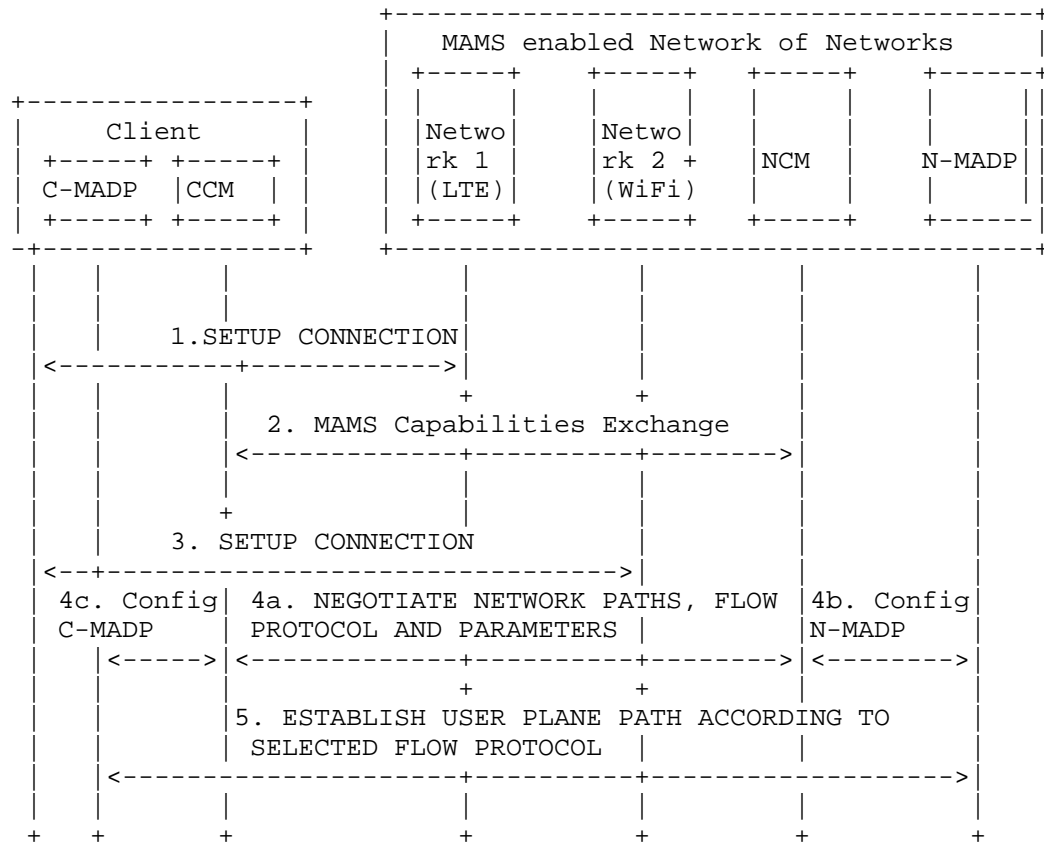


Figure 2: MAMS call flow

Figure 2 illustrates the MAMS signaling mechanism for negotiation of network paths and flow protocols between the client and the network. In this example scenario, the client is connected to two networks (say LTE and WiFi).

1. UE connects to network 1 and gets an IP address assigned by network 1.
2. CCM communicates with NCM functional element via the network 1 connection and exchanges capabilities and parameters for MAMS operation. Note: The NCM credentials can be made known to the UE by pre-provisioning.

3. Client sets up connection with network 2 and gets an IP address assigned by network 2.
4. CCM and NCM negotiate capabilities and parameters for establishment of network paths, which are then used to configure user plane functions N-MADP at the network and C-MADP at the client.
 - 4a. CCM and NCM negotiate network paths, flow routing and aggregation protocols, and related parameters.
 - 4b. NCM communicates with the N-MADP to exchange and configure flow aggregation protocols, policies and parameters in alignment with those negotiated with the CCM.
 - 4c. CCM communicates with the C-MADP to exchange and configure flow aggregation protocols, policies and parameters in alignment with those negotiated with the NCM.
5. C-MADP and N-MADP establish the user plane paths, e.g. using IKE [RFC7296] signaling, based on the negotiated flow aggregation protocol and parameters specified by NCM.

CCM and NCM can further exchange messages containing access link measurements for link maintenance by the NCM. NCM evaluates the link conditions in the UL and DL across LTE and WiFi, based on link measurements reported by CCM and/or link probing techniques and determines the UL and DL user data distribution policy. NCM and CCM also negotiate application level policies for categorizing applications, e.g. based on DSCP, Destination IP address, and determining which of the available network paths, needs to be used for transporting data of that category of applications. NCM configures N-MADP and CCM configures C-MADP based on the negotiated application policies. CCM may apply local application policies, in addition to the application policy conveyed by the NCM.

9. Implementation considerations

MAMS builds on commonly available functions available on terminal devices that can be delivered as a software update over the popular end-user device operating systems, enabling rapid deployment and addressing the large deployed device base.

10. Applicability to Mobile Edge Computing

Mobile edge computing (MEC) is an access-edge cloud platform being standardized at ETSI, whose initial focus was to improve quality of experience by leveraging intelligence at cellular (e.g. 3GPP

technologies like LTE) access edge, and the scope is now being extended to support access technologies beyond 3GPP. This applicability of the framework described in this document to the MEC platform has been evaluated and tested in different network configurations.

The NCM and N-MADP are hosted on the MEC cloud server that is located in the user plane path at the edge of multi-technology access networks, and in a particular large venue use case at the edge of LTE and Wi-Fi access networks. The NCM and CCM negotiate the network path combinations based on application needs and the necessary user plane protocols to manage the multiple paths. The network conditions reported by the CCM to the NCM is used in addition to Radio Analytics application residing at the MEC to configure the uplink and downlink access paths according to changing radio and congestion conditions.

The aim of these enhancements is to improve the end-user's quality of experience by leveraging the best network path based on application needs and network conditions.

11. Security Considerations

This section details the security considerations for the MAMS framework.

11.1. Data and Control plane security

Signaling messages and the user data in MAMS framework rely on the security of the underlying network transport paths. When this cannot be assumed, network connection manager configures use of protocols, like IPsec [RFC4301] [RFC3948], for securing user data and MAMS signaling messages.

12. Contributors

This protocol is the outcome of work by many engineers, not just the authors of this document. In alphabetical order, the contributors to the project are: Barbara Orlandi, Bongho Kim, David Lopez-Perez, Doru Calin, Jonathan Ling, Krishna Pramod A., Lohith Nayak, Michael Scharf.

13. References

13.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <<http://www.rfc-editor.org/info/rfc4301>>.

13.2. Informative References

- [RFC3948] Huttunen, A., Swander, B., Volpe, V., DiBurro, L., and M. Stenberg, "UDP Encapsulation of IPsec ESP Packets", RFC 3948, DOI 10.17487/RFC3948, January 2005, <<http://www.rfc-editor.org/info/rfc3948>>.
- [RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <<http://www.rfc-editor.org/info/rfc7296>>.

Authors' Addresses

Satish Kanugovi
Nokia

Email: satish.k@nokia.com

Subramanian Vasudevan
Nokia

Email: vasu.vasudevan@nokia.com

Florin Baboescu
Broadcom

Email: florin.baboescu@broadcom.com

Jing Zhu
Intel

Email: jing.z.zhu@intel.com

INTAREA
Internet-Draft
Intended status: Standards Track
Expires: April 29, 2017

E. Nordmark
Arista Networks
October 26, 2016

IP over Intentionally Partially Partitioned Links
draft-nordmark-intarea-ippl-05

Abstract

IP makes certain assumptions about the L2 forwarding behavior of a multi-access IP link. However, there are several forms of intentional partitioning of links ranging from split-horizon to Private VLANs that violate some of those assumptions. This document specifies that link behavior and how IP handles links with those properties.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 29, 2017.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Keywords and Terminology	3
3. Private VLAN	4
3.1. Bridge Behavior	4
4. IP over IPPL	5
5. IPv6 over IPPL	5
6. IPv4 over IPPL	6
7. Multiple routers	7
8. Multicast over IPPL	8
9. DHCP Implications	8
10. Redirect Implications	9
11. Security Considerations	9
12. IANA Considerations	9
13. Acknowledgements	9
14. Appendix: Layer 2 Implications	9
15. References	10
15.1. Normative References	10
15.2. Informative References	10
Author's Address	12

1. Introduction

IPv4 and IPv6 can in general handle two forms of links; point-to-point links when only have two IP nodes (self and remote), and multi-access links with one or more nodes attached to the link. For the multi-access links IP in general, and particular protocols like ARP and IPv6 Neighbor Discovery, makes a few assumptions about transitive and reflexive connectivity i.e., that all nodes attached to the link can send packets to all other nodes.

There are cases where for various reasons and deployments one wants what looks like one link from the perspective of IP and routing, yet the L2 connectivity is restrictive. A key property is that an IP subnet prefix is assigned to the link, and IP routing sees it as a regular multi-access link. But a host attached to the link might not be able to send packets to all other hosts attached to the link. The motivation for this is outside the scope of this document, but in summary the motivation to preserve the subnet view as seen by IP routing is to conserve IP(v4) address space, and the motivation to restrict communication on the link could be due to (security) policy or potentially wireless connectivity approaches.

This intentional and partial partition appears in a few different forms. For DSL [TR-101] and Cable [DOCSIS-MULPI] the pattern is to have a single access router on the link, and all the hosts can send and receive from the access router, but host-to-host communication is blocked. A richer set of restrictions are possible for Private VLANs (PVLAN) [RFC5517], which has a notion of three different ports i.e. attachment points: isolated, community, and promiscuous. Note that other techniques operate at L2/L3 boundary like [RFC4562] but those are out of scope for this document.

The possible connectivity patterns for PVLAN appears to be a superset of the DSL and Cable use of split horizon, thus this document specifies the PVLAN behavior, shows the impact on IP/ARP/ND, and specifies how IP/ARP/ND must operate to work with PVLAN.

If private VLANs, or the split horizon subset, has been configured at layer 2 for the purposes of IPv4 address conservation, then that layer 2 configuration will affect IPv6 even though IPv6 might not have the same need for address conservation.

2. Keywords and Terminology

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in [RFC2119].

The following terms from [RFC4861] are used without modifications:

node	a device that implements IP.
router	a node that forwards IP packets not explicitly addressed to itself.
host	any node that is not a router.
link	a communication facility or medium over which nodes can communicate at the link layer, i.e., the layer immediately below IP. Examples are Ethernets (simple or bridged), PPP links, X.25, Frame Relay, or ATM networks as well as Internet-layer (or higher-layer) "tunnels", such as tunnels over IPv4 or IPv6 itself.
interface	a node's attachment to a link.
neighbors	nodes attached to the same link.

This document defines the following set of terms:

bridge	a layer-2 device which implements 802.1Q
port	a bridge's attachment to another bridge or to a node.

3. Private VLAN

A private VLAN is a structure which uses two or more 802.1Q (VLAN) values to separate what would otherwise be a single VLAN, viewed by IP as a single broadcast domain, into different types of ports with different L2 forwarding behavior between the different ports. A private VLAN consists of a single primary VLAN and multiple secondary VLANs.

From the perspective of both a single bridge and a collection of interconnected bridges there are three different types of ports used to attach nodes plus an inter-bridge port:

- o Promiscuous: A promiscuous port can send packets to all ports that are part of the private VLAN. Such packets are sent using the primary VLAN ID.
- o Isolated: Isolated VLAN ports can only send packets to promiscuous ports. Such packets are sent using an isolated VLAN ID.
- o Community: A community port is associated with a per-community VLAN ID, and can send packets to both ports in the same community VLAN and promiscuous ports.
- o Inter-bridge: A port used to connect a bridge to another bridge.

3.1. Bridge Behavior

Once a bridge or a set of interconnected bridges have been configured with both the primary and isolated VLAN ID, and zero or more community VLAN IDs associated with the private VLAN, the following forward behaviors apply to the bridge:

- o A packet received on an isolated port MUST NOT be forwarded out an isolated or community port; it SHOULD (subject to bandwidth/resource issues) be forwarded out promiscuous and inter-bridge ports.
- o A packet received on a community port MUST NOT be forwarded out an isolated port or a community port with a different VLAN ID; it SHOULD be forwarded out promiscuous and inter-bridge ports as well as community ports that have the same community VLAN ID.
- o A packet received on a promiscuous port SHOULD be forwarded out all types of ports in the private VLAN.
- o A packet received on an inter-bridge port with an isolated VLAN ID should be forwarded as a packet received on an isolated port.
- o A packet received on an inter-bridge port with a community VLAN ID should be forwarded as a packet received on a community port associated with that VLAN ID.
- o A packet received on an inter-bridge port with a promiscuous VLAN ID should be forwarded as a packet received on a promiscuous port.

In addition to the above VLAN filtering and implied MAC address learning rules, the packet forwarding is also subject to the normal 802.1Q rules with blocking ports due to spanning-tree protocol etc.

4. IP over IPPL

When IP is used over Intentionally Partially Partitioned links like private VLANs the normal usage is to attached routers (and potentially other shared resources like servers) to promiscuous ports, while attaching other hosts to either community or isolated ports. If there is a single host for a given tenant or other domain of separation, then it is most efficient to attach that host to an isolated port. If there are multiple hosts in the private VLAN that should be able to communicate at layer 2, then they should be assigned a common community VLAN ID and attached to ports with that VLAN ID.

The above configuration means that hosts will not be able to communicate with each other unless they are in the same community. However, mechanisms outside of the scope of this document can be used to allow IP communication between such hosts e.g., by having firewall or gateway in or beyond the routers connected to the promiscuous ports. When such a policy is in place it is important that all packets which cross communities are sent to a router, which can have access-control lists or deeper firewall rules to decide which packets to forward.

5. IPv6 over IPPL

IPv6 Neighbor Discovery [RFC4861] can be used to get all the hosts on the link to send all unicast packets except those send to link-local destination addresses to the routers. That is done by setting the L-flag (on-link) to zero for all of the Prefix Information options. Note that this is orthogonal to whether SLAAC (Stateless Address Auto-Configuration) [RFC4862] or DHCPv6 [RFC3315] is used for address autoconfiguration. Setting the L-flag to zero is RECOMMENDED configuration for private VLANs.

If the policy includes allowing some packets that are sent to link-local destinations to cross between different tenants, then some form of NS/NA proxy is needed in the routers, and the routers need to forward packets addressed to link-local destinations out the same interface as REQUIRED in [RFC2460]. If the policy allows for some packets sent to global IPv6 address to cross between tenants then the routers would forward such packets out the same interface. However, with the L=0 setting those global packets will be sent to the default router, while the link-local destinations would result in a Neighbor Solicitation to resolve the IPv6 to link-layer address binding.

Handling such a NS when there are multiple promiscuous ports hence multiple routers risks creating loops. If the router already has a neighbor cache entry for the destination it can respond with an NA on behalf of the destination. However, if it does not it MUST NOT send a NS on the link, since the NA will be received by the other router(s) on the link which can cause an unbounded flood of multicast NS packets (all with hoplimit 255), in particular of the host IPv6 address does not respond. Note that such an NS/NA proxy is defined in [RFC4389] under some topological assumptions such as there being a distinct upstream and downstream direction, which is not the case of two or more peer routers on the same IPPL. For that reason NS/NA packet proxies as in [RFC4389] MUST NOT be used with IPPL.

IPv6 includes Duplicate Address Detection [RFC4862], which assumes that a link-local IPv6 multicast can be received by all hosts which share the same subnet prefix. That is not the case in a private VLAN, hence there could potentially be undetected duplicate IPv6 addresses. However, the DAD proxy approach [RFC6957] defined for split-horizon behavior can safely be used even when there are multiple promiscuous ports hence multiple routers attached to the link, since it does not rely on sending Neighbor Solicitations instead merely gathers state from received packets. The use of [RFC6957] with private VLAN is RECOMMENDED.

The Router Advertisements in a private VLAN MUST be sent out on a promiscuous VLAN ID so that all nodes on the link receive them.

6. IPv4 over IPPL

IPv4 [RFC0791] and ARP [RFC0826] do not have a counterpart to the Neighbor Discovery On-link flag. Hence nodes attached to isolated or community ports will always ARP for any destination which is part of its configured subnet prefix, and those ARP request packets will not be forwarded by the bridges to the target nodes. Thus the routers attached to the promiscuous ports MUST provide a robust proxy ARP mechanism if they are to allow any (firewalled) communication between nodes from different tenants or separation domains.

For the ARP proxy to be robust it MUST avoid loops where router1 attached to the link sends an ARP request which is received by router2 (also attached to the link), resulting in an ARP request from router2 to be received by router1. Likewise, it MUST avoid a similar loop involving IP packets, where the reception of an IP packet results in sending a ARP request from router1 which is proxied by router2. At a minimum, the reception of an ARP request MUST NOT result in sending an ARP request, and the routers MUST either be configured to know each others MAC addresses, or receive the VLAN tagged packets so they can avoid proxying when the packet is received

on with the promiscuous VLAN ID. Note that should there be an IP forwarding loop due to proxying back and forth, the IP TTL will expire avoiding unlimited loops.

Any proxy ARP approach MUST work correctly with Address Conflict Detection [RFC5227]. ACD depends on ARP probes only receiving responses if there is a duplicate IP address, thus the ARP probes MUST NOT be proxied. These ARP probes have a Sender Protocol Address of zero, hence they are easy to identify.

When proxying an ARP request (with a non-zero Sender Protocol Address) the router needs to respond by placing its own MAC address in the Sender Hardware Address field. When there are multiple routers attached to the private VLAN this will not only result in multiple ARP replies for each ARP request, those replies would have a different Sender Hardware Address. That might seem surprising to the requesting node, but does not cause an issue with ARP implementations that follow the pseudo-code in [RFC0826].

If the two or more routers attached to the private VLAN implement VRRP [RFC5798] the routers MAY use their VRRP MAC address as the Sender Hardware Address in the proxied ARP replies, since this reduces the risk nodes that do not follow the pseudo-code in [RFC0826]. However, if they do so it can cause flapping of the MAC tables in the bridges between the routers and the ARPing node. Thus such use is NOT RECOMMENDED in general topologies of bridges but can be used when there are no intervening bridges.

7. Multiple routers

In addition to the above issues when multiple routers are attached to the same PVLAN, the routers need to avoid potential routing loops for packets entering the subnet. When such a packet arrives the router might need to send a ARP request (or Neighbor Solicitation) for the host, which can trigger the other router to send a proxy ARP (or Neighbor Advertisement). The host, if present, will also respond to the ARP/NS. This issue is described in [PVLAN-HOSTING] in the particular case of HSRP.

When multiple routers are attached to the same PVLAN, wheter they are using VRRP, HSRP, or neither, they SHOULD NOT proxy ARP/ND respond to a request from another router. At a minimum a router MUST be configurable with a list of IP addresses to which it should not proxy respond. Thus the user can configure that list with the IP address(es) of the other router(s) attached to the PVLAN.

8. Multicast over IPPL

Layer 2 multicast or broadcast is used by protocols like ARP [RFC0826], IPv6 Neighbor Discovery [RFC4861] and Multicast DNS [RFC6762] with link-local scope. The first two have been discussed above.

Multicast DNS can be handled by implementing using some proxy such as [I-D.ietf-dnssd-hybrid] but that is outside of the scope of this document.

IP Multicast which spans across multiple IP links and that have senders that are on community or isolated ports require additional forwarding mechanisms in the routers that are attached to the promiscuous ports, since the routers need to forward such packets out to any allowed receivers in the private VLAN without resulting in packet duplication. For multicast senders on isolated ports such forwarding would result in the sender potentially receiving the packet it transmitted. For multicast senders on community ports, any receivers in the same community VLAN are subject to receiving duplicate packets; one copy directly from layer 2 from the sender and a second copy forwarded by the multicast router.

For that reason it is NOT RECOMMENDED to configure outbound multicast forwarding from private VLANs.

9. DHCP Implications

With IPv4 both a static configuration and a DHCPv4 configuration will assign a subnet prefix to any hosts including those attached to the isolated or community ports. Hence the above robust proxy ARP is needed even in the case of DHCPv4.

With IPv6 static configuration, or SLAAC (Stateless Address Auto-Configuration) [RFC4862] or DHCPv6 [RFC3315] can be used to configure the IPv6 addresses on the interfaces. However, when DHCPv6 is used to configure the IPv6 addresses it does not configure any notion of an on-link prefix length. Thus in that case the on-link determination comes from the Router Advertisement. Hence the above approach of setting L=0 in the Prefix Information Option will result in packets being sent to the default router(s).

Hence no special considerations are needed for DHCPv4 or DHCPv6.

10. Redirect Implications

ICMP redirects can be used for both IPv4 and IPv6 to indicate a better first-hop router to hosts, and in addition for IPv6 can be used to indicate the direct link-layer address to use to send to a node which is on the link. ICMP redirects to another router which attached to a promiscuous port would work since the host can reach it. However, communication will fail if that port is not promiscuous. In addition, the IPv6 redirect to an on-link host is likely to be problematic since a host is likely to be attached to an isolated or community port.

For those reasons it is RECOMMENDED that the sending of IPv4 and IPv6 redirects is disabled on the routers attached to the IPPL.

11. Security Considerations

In general DAD is subject to a Denial of Service attack since a malicious host can claim all the IPv6 addresses [RFC3756]. Same issue applies to IPv4/ARP when Address Conflict Detection [RFC5227] is implemented.

12. IANA Considerations

There are no IANA actions needed for this document.

13. Acknowledgements

The author is grateful for the comments from Mikael Abrahamsson, Fred Baker, Wes Beebe, Hemant Singh, Dave Thaler, and Sowmini Varadhan.

14. Appendix: Layer 2 Implications

While not in scope for this document, there are some observations relating to the interaction of IPPL (and private VLANs in particular) and layer 2 learning which are worth mentioning. Depending on the details of how the deployed Ethernet bridges perform learning, a side effect of using a different .1Q tag for packets sent from the routers than for packets sent towards the routers mean that the 802.1Q learning and aging process in intermediate bridges might age out the MAC address entry for the routers MAC address. If that happens packets sent towards the router will be flooded at layer two. The observed behavior is that an ARP request for the router's IP address will result in re-learning the MAC address. Thus some operators work around this issue by configuring the ARP aging time to be shorter than the MAC aging time.

15. References

15.1. Normative References

- [RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <<http://www.rfc-editor.org/info/rfc791>>.
- [RFC0826] Plummer, D., "Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware", STD 37, RFC 826, DOI 10.17487/RFC0826, November 1982, <<http://www.rfc-editor.org/info/rfc826>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460, December 1998, <<http://www.rfc-editor.org/info/rfc2460>>.
- [RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, DOI 10.17487/RFC4861, September 2007, <<http://www.rfc-editor.org/info/rfc4861>>.
- [RFC4862] Thomson, S., Narten, T., and T. Jinmei, "IPv6 Stateless Address Autoconfiguration", RFC 4862, DOI 10.17487/RFC4862, September 2007, <<http://www.rfc-editor.org/info/rfc4862>>.
- [RFC6957] Costa, F., Combes, J-M., Ed., Pournard, X., and H. Li, "Duplicate Address Detection Proxy", RFC 6957, DOI 10.17487/RFC6957, June 2013, <<http://www.rfc-editor.org/info/rfc6957>>.

15.2. Informative References

- [DOCSIS-MULPI]
"DOCSIS 3.0: MAC and Upper Layer Protocols Interface Specification", August 2015, <<http://www.cablelabs.com/wp-content/uploads/specdocs/CM-SP-MULPIv3.0-I28-150827.pdf>>.

- [I-D.ietf-dnssd-hybrid]
Cheshire, S., "Hybrid Unicast/Multicast DNS-Based Service Discovery", draft-ietf-dnssd-hybrid-03 (work in progress), February 2016.
- [PVLAN-HOSTING]
"PVLANS in a Hosting Environment", March 2010,
<<https://puck.nether.net/pipermail/cisco-nsp/2010-March/068469.html>>.
- [RFC3315] Droms, R., Ed., Bound, J., Volz, B., Lemon, T., Perkins, C., and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3315, DOI 10.17487/RFC3315, July 2003, <<http://www.rfc-editor.org/info/rfc3315>>.
- [RFC3756] Nikander, P., Ed., Kempf, J., and E. Nordmark, "IPv6 Neighbor Discovery (ND) Trust Models and Threats", RFC 3756, DOI 10.17487/RFC3756, May 2004, <<http://www.rfc-editor.org/info/rfc3756>>.
- [RFC4389] Thaler, D., Talwar, M., and C. Patel, "Neighbor Discovery Proxies (ND Proxy)", RFC 4389, DOI 10.17487/RFC4389, April 2006, <<http://www.rfc-editor.org/info/rfc4389>>.
- [RFC4562] Melsen, T. and S. Blake, "MAC-Forced Forwarding: A Method for Subscriber Separation on an Ethernet Access Network", RFC 4562, DOI 10.17487/RFC4562, June 2006, <<http://www.rfc-editor.org/info/rfc4562>>.
- [RFC5227] Cheshire, S., "IPv4 Address Conflict Detection", RFC 5227, DOI 10.17487/RFC5227, July 2008, <<http://www.rfc-editor.org/info/rfc5227>>.
- [RFC5517] HomChaudhuri, S. and M. Foschiano, "Cisco Systems' Private VLANs: Scalable Security in a Multi-Client Environment", RFC 5517, DOI 10.17487/RFC5517, February 2010, <<http://www.rfc-editor.org/info/rfc5517>>.
- [RFC5798] Nadas, S., Ed., "Virtual Router Redundancy Protocol (VRRP) Version 3 for IPv4 and IPv6", RFC 5798, DOI 10.17487/RFC5798, March 2010, <<http://www.rfc-editor.org/info/rfc5798>>.
- [RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, DOI 10.17487/RFC6762, February 2013, <<http://www.rfc-editor.org/info/rfc6762>>.

[TR-101] "Migration to Ethernet-Based DSL Aggregation", The
Broadband Forum Technical Report TR-101, July 2011,
<[http://www.broadband-forum.org/technical/download/
TR-101_Issue-2.pdf](http://www.broadband-forum.org/technical/download/TR-101_Issue-2.pdf)>.

Author's Address

Erik Nordmark
Arista Networks
Santa Clara, CA
USA

Email: nordmark@arista.com

Network Working Group
Internet-Draft
Intended status: BCP

L. Yong
L. Dunbar
Huawei
T. Herbert
Facebook

Expires: April 2017

October 31, 2016

Interconnecting Network Sites by IP Tunnels
draft-yong-intarea-inter-sites-over-tunnels-00.txt

Abstract

This document specifies use of a set of IP tunnels to interconnect multiple network sites over IP backbone networks. The interconnected network sites form 'virtual' private network(s). The networks at any of those sites can be Layer 2 domains or/and Layer 3 subnets. The IP backbone networks that the tunnels traverse through can be IPv4 or IPv6. This document describes the special property (or features) that those IP tunnels need to have in order to interconnect multiple sites as if those sites are directly connected by wires.

Status of This Document

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 31, 2017.

Copyright Notice

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

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

Table of Contents

1. Introduction.....	3
1.1. Motivation of sites interconnection by IP tunnels.....	3
2. Terminology.....	4
2.1. Requirements language.....	4
2.2. Terms defined in this document.....	5
3. Sites interconnection solution architecture.....	5
4. Key properties of sites interconnection over IP tunnels.....	7
4.1. Network sites interconnection properties.....	7
4.2. Tunnel transport properties for sites interconnection.....	8
5. Site traffic encapsulation.....	9
6. Tunneling multicast and broadcast traffic.....	9
7. Tunnel transport over IP.....	9
7.1. Tunnel transport mode.....	9
7.2. MTU and fragmentation.....	10
7.3. Checksum.....	10
7.4. Congestion management.....	10
7.4.1. Congestion detection.....	10
7.4.2. Congestion notification.....	10
7.4.3. Tunnel ingress traffic control.....	10
7.5. Tunnel traffic hop count and DSCP value setting.....	10
7.6. Middle box considerations.....	10
8. Sites interconnection security.....	10
9. Tunnel configuration.....	11
10. Tunnel tools.....	11
11. Operational considerations.....	11
12. IANA considerations.....	11
13. Security considerations.....	11
14. References.....	11
14.1. Normative References.....	11
14.2. Informative Reference.....	11
15. Authors' Addresses.....	12

1. Introduction

This document specifies use of a set of IP tunnels to interconnect multiple network sites over IP backbone networks. The networks at any of those sites can be Layer 2 domains or Layer 3 subnets; IP backbone networks that tunnels traverse can be IPv4 or IPv6. This document describes the properties (or features) that IP tunnels need to have in order to interconnect multiple sites as if those sites are directly connected by wires.

An IP tunnel in this document is identified by a pair of IP addresses (IPv4 or IPv6) that IP backbone networks can reach or the IP addresses plus a pair of UDP ports; one address per each tunnel endpoint. Tunnel ingress receives network traffic from its site (or access ports) and will encapsulate the traffic with an outer header whose destination and source address are the tunnel's two end points respectively. Tunnel egress receives IP packets from the backbone network, decapsulates the IP packets, and forwards decapsulated traffic toward its site (or an access port).

Section 1.1 describes the motivation of this specification. Section 3 describes the solution architecture for sites interconnection by IP tunnels. Section 4 specifies the sites interconnection requirements for the IP tunnels. The rest of Sections describe Site Interconnection Tunnel (SITE) solution. Section 11 describes operational considerations for sites interconnection over IP tunnels.

Note: The site interconnection policy configuration and the prefix routing within a site and across sites are outside the scope of this document.

1.1. Motivation of sites interconnection by IP tunnels

Tunneling technology has being widely used in IP networks. For examples: transporting packets in one address family (e.g. IPv6) over a network of different address family (e.g. IPv4) [RFC7059]; establishing a direct logical "link" for traffic engineering; traffic security over the Internet (e.g. IPsec tunnel [RFC5996] [RFC3884]); or Location and Identifier Separation application (LISP) [RFC6830].

Provider based L2VPN and L3VPN solutions [RFC4762] [RFC4364] use MPLS LSP tunnel technology to interconnect provider edge devices, i.e., Provider Edge (PE). In these solutions, PEs are part of provider backbone networks that implement the VPN solutions. For a customer to get a provider based VPN service, each customer site

must attach to at least one of the PEs in the provider backbone networks, this attachment is called attachment circuit (AC) [RFC4664] [RFC4364].

Today, a company can use IPSec tunnels [RFC5996][RFC3884] to interconnect its IP subnets at different sites over the Internet if the company has an experienced operator to configure its site networks and the devices that support the IPsec tunnel capability. To achieve this, each site needs to have at least one device attaching to an IP backbone network and have at least one public IP address that the IP backbone networks can reach to. As a result, the interconnected sites form one "virtual" private network among the sites. In this case, an ISP provider (i.e. IP backbone provider) does not provide the sites interconnection service to the company although it carries the IPsec tunnel traffic; in other words, the ISP provider only provides Internet access service to each site. As a result, there is no guaranteed QoS between a pair of sites, which is the difference from provider based VPN solutions.

Sites interconnection by IP tunnels is attractive to some companies that operate at multiple locations. Some companies wish to do it but do not have such an experienced operator to construct/configure site networks and tunnels to achieve this. A vendor may make a product to achieve this and sell to these companies. The product includes a site gateway device or component (called S-GW in this document) and software to allow a customer to specify their site interconnection requirements including sites interconnection policies. The software will manage the configuration of the S-GWs at multiple locations; furthermore the software can automate the processes from client specifying the sites interconnection to the sites interconnection completion. Section 3 highlights this solution architecture.

A SP provider may use this product and integrate it with its provider based VPN solution [RFC4364][RFC4762][RFC7432] to provide agile VPN services to its customers. [Dunbar]

2. Terminology

2.1. Requirements language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

- o Site Gateway (S-GW): A router and/or switch device/component. It can be configured to perform routing/forwarding function at the site. S-GW attaches to an IP backbone network and support a tunnel capability specified in this document. S-GW can be configured with one IP address to be used as tunnel end point for all tunnels to other sites or be configured with one IP address per a tunnel to a remote site.
- o Sites Interconnection Controller (SIC): A software component that controls/manages individual S-GWs via a SIC channel. It receives a sites interconnection request from the SIP; processes it, and sends the configuration data to individual S-GWs via SIC channel.
- o Sites Interconnection Portal: A software with GUI interface to allow site operator to specify its sites interconnection requirement including sites interconnection policy. The same portal can be used for the site operator to specify the site Internet access policies.
- o SIC Channel: a communication channel between S-GW and SIC over IP network. It could be a TCP application with security capability. The channel is used for SIC to send S-GW configuration data and get the PM data from S-GW. It may be used for dynamic routing purpose among the sites; in this case, SIC gets the prefix information from the S-GW at a site, and send the information to the S-GWs at other sites where the S-GW will distribute the information to the site.
- o IP tunnel: A tunnel exists between a pair of S-GWs. The tunnel end point as an interface on a S-GW receives network traffic and encapsulate the traffic with an outer header whose Destination and Source address are the tunnel's two end points respectively. Tunnel egress receives IP packets from the IP backbone network, decapsulates the IP packets, and forwards decapsulated packets toward its site (or one access port(s)). IP tunnel is unidirectional, two sites interconnection requires two tunnels, one for each direction. Two tunnels may traverse different paths in backbone network.

In this architecture, each site uses its S-GW to attach to IP backbone network, which implies that the site traffic may go to the Internet via the S-GW. For security and performance reason, it is RECOMMENDED to use different public IP addresses for the Internet access and the tunnel end point of sites interconnection. It is possible to use more than one S-GW at a site for sites interconnection.

In this architecture, a site may have L2 domains, L3 subnets, or TRILL networks. However, the interconnected site networks **MUST** have the same type (see Section 3.1).

Note: This document will not specify the whole solution for this architecture. It only specifies the tunnel end point functions at a S-GW and tunnel end point configuration at the S-GW. Other functions in this solution architecture will be addressed in other documents.

4. Key properties of sites interconnection over IP tunnels

This section only lists the requirements for sites interconnection over IP tunnels. The entire solution architecture requirements are beyond the scope of this document.

4.1. Network sites interconnection properties

1. A site in this context may have a single network or multiple networks. For single network case, the network may be an L2 domain, L3 subnets, or a TRILL network. For the case of multiple networks, each network may be an L2 domain or L3 subnets and individual network traffic is segregated within the site including on the S-GW. Sites interconnection **MUST NOT** interconnect the networks at two sites that have different type. For example, one is L3 subnet, another is L2 domain. In the case of multiple networks at a site, sites interconnection **MUST** support individual networks at the site to be connected to the same type of networks that reside in either a remote site or different remote sites.
2. The S-GW **MUST** support site Internet access. The traffic to the Internet and the traffic to a remote site **SHOULD** be segregated by different S-GW physical ports or different IP addresses to avoid DDoS attack [RFC4732].
3. Tunnels for sites interconnection **MUST** secure site traffic over the IP backbone network.
4. Tunnels for sites interconnection **MUST** segregate the site traffic from different networks.
5. Sites interconnection topology may be mesh or hub-spoke.

6. Tunnel for sites interconnection MUST be able to detect congestion on the path of IP backbone network and MUST stop some site traffic based on the operator specified policy. (stop some traffic on both directions or congestion direction?). Early dropping traffic will help site applications to take an action. S-GW MUST reports the congestion status to site interconnection app (SIA).
7. A site can be configured with two S-GWs for redundant and/or transport capacity. The remote S-GW MUST be able to support load balance tunnel traffic. A S-GW at a site MUST NOT forward incoming traffic from IP backbone to another S-GW at the site, which creates the loop.
8. Site network traffic may be unicast, mcast, or bcast(L2) traffic. Sites interconnection solution MUST be able to carry unicast, mcast, bcast traffic over tunnels.
9. Site network traffic may be control plane packets such as IBGP (ISIS?) or control packets such as ICMP, ARP. Tunnel SHOULD be able to carry control plane or control packets according to operator specified site interconnection policy. In default, a tunnel MUST NOT carry control plane packets over the tunnel.
10. Other?

4.2. Tunnel transport properties for sites interconnection

This section lists tunnel transport requirements over IP backbone networks for sites interconnection application.

1. Tunnel type (tunnel mode, or transport mode, or both)
2. IP backbone network can be IPv4 or IPv6 network. A tunnel MUST be able to run over an IPv4 or IPv6 network.
3. Tunnel solution MUST meet IP [RFC791] [RFC2460] and UDP application requirements [RFC5405bis]
4. Tunnel MTU and Fragmentation [JOE]. Tunnel SHOULD avoid tunnel traffic to be fragmented on IP backbone networks.
5. Tunnel solution supports configuration option to turn off traffic encryption function.

6. Tunnel solution supports congestion control upon detecting congestion condition on the tunnel path (use circuit breaker, packet drop/report at ingress, notification to source).
 7. Tunnel solution MUST map site traffic QoS to proper DSCP value on tunnel IP packets based on operation specified policy.
 8. Tunnel solution SHOULD be able to monitor the inter-sites connectivity and report the status.
 9. Tunnel solution SHOULD support some tools for sites interconnection operator such as tunnel path trace, ping, tunnel's throughput test, etc.
 10. ICMP handling (from Internet or from remote tunnel end-point).
 11. Others, Hop count for tunnel traffic, middle box considerations, etc.
5. Site traffic encapsulation
- GRE-in-UDP w/DTLS [GRE-in-UDP] or GUE [GUE]: the reason is:
- o It can run over the Internet (Ipv4 and Ipv6)
 - o It runs as UDP application, ECMP advantage, and middle box traversal benefit.
 - o Encapsulate different types of traffic
 - o Ability to support fragmentation
 - o Security/encryption capability
 - o Support traffic segregation
- Like to hear other's opinions on encapsulation protocol choices.
6. Tunneling multicast and broadcast traffic
7. Tunnel transport over IP
- 7.1. Tunnel transport mode
- Tunnel mode or transport mode or both.[RFC3884]

7.2. MTU and fragmentation

Intarea-tunnels draft or GUE-extension draft

7.3. Checksum

Tunnel solution MUST implement the checksum specification for default GRE-in-UDP tunnel in [GRE-in-UDP].

7.4. Congestion management

More than likely, a site operator has no way to control transport path resource in IP backbone networks for sites interconnection. Tunnel packets are treated as regular IP packets and traverse a path in IP backbone networks that other IP application packets traverse as well. Congestion may happen due to "ship-in-night" situation. IP backbone network uses explicitly congestion notification (ECN) [RFC6040] to indicate IP applications about the network congestion.

Upon backbone path congestion, a tunnel for the site interconnection MUST stop some traffic based on operator's interconnection policy. This section specifies the tunnel congestion control mechanism.

7.4.1. Congestion detection

7.4.2. Congestion notification

7.4.3. Tunnel ingress traffic control

7.5. Tunnel traffic hop count and DSCP value setting

7.6. Middle box considerations

8. Sites interconnection security

For site traffic security, tunnel MUST use DTLS to encrypt site traffic, i.e., use GRE-in-UDP w/ DTLS. This feature MAY be turned off by a site operator if the site network traffic is already encrypted.

A site operator SHOULD request a security service from IP backbone provider to prevent DDoS traffic to reach the tunnel end point at a S-GW of the sites.

9. Tunnel configuration

10. Tunnel tools

11. Operational considerations

12. IANA considerations

13. Security considerations

14. References

14.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC2119, March 1997.
- [RFC791] DARPA, "INTERNET PROTOCOL", RFC791, September, 1981.
- [RFC792] Postel, J., "Internet Control Message Protocol", STD 5, RFC792, September 1981.
- [RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.
- [RFC5405bis] Eggert, L., "Unicast UDP Usage Guideline for Application Designers", draft-ietf-tsvwg-rfc5405bis, work in progress.
- [RFC6040] Briscoe, B., "Tunneling of Explicit Congestion Notification", RFC6040, November 2010.
- [GRE-in-UDP] Yong, L., et al, "GRE-in-UDP Encapsulation", draft-ietf-tsvwg-gre-in-udp-encap-19, work in progress.
- [JOE] Touch, J. and Townsley, M., "IP Tunnels in the Internet Architecture", draft-ietf-intarea-tunnels-03, work in progress.

14.2. Informative Reference

- [RFC3884] Touch, J., Eggert, L., and Wang, Y., "Use of Ipsec Transport Mode for Dynamic Routing", September 2004.

- [RFC4364] Rosen, E. and Rekhter, Y., "BGP/MPLS IP Virtual Private Networks (VPNs)", RFC4364, February 2006.
- [RFC4664] Andersson, L. and Rosen, E., "Framework for Layer 2 Virtual Private Networks (L2VPNs)", RFC4664, September 2006.
- [RFC4732] Handley, M. and Rescorla, E., "Internet Denial-of-Service Considerations" RFC4732, November 2006.
- [RFC4762] Lasserre, M. and Kompella, V., "Virtual Private LAN Service (VPLS) using Label Distribution Protocol (LDP) Signaling", RFC4762, January 2007.
- [RFC5996] Kaufman, C., et al, "Internet Key Exchange Protocol Version 2 (IKEv2)", September 2010.
- [RFC6830] Farinacci, D., et al, "The Locator/IP Separation Protocol (LISP)", RFC6830, January 2013.
- [RFC7059] Steffann, S., et al, "A comparison of IPv6-over-IPv4 Tunnel Mechanism", RFC7059, November 2013
- [RFC7432] Sajassi, A., et al, "BGP MPLS-Based Ethernet VPN", RFC7432, February 2015.
- [Dunbar] Dunbar, L., Yong, L., Song, X., "Client Defined Private Networks laid over Thin CEPs", draft-dunbar-interarea-private-networks-over-thinCPE, work in progress.
- [GUE] Herbert, T., Yong, L., Zia, O, "Generic UDP Encapsulation", draft-ietf-nvo3-gue-05, work in progress.

15. Authors' Addresses

Lucy Yong
Huawei Technologies

Email: lucy.yong@huawei.com

Linda Dunbar
Huawei Technologies

Email: linda.dunbar@huawei.com

Tom Herbert

Facebook

Emails: tom@herbertland.com

