INTAREA
Internet-Draft
Updates: RFC 4884 (if approved)
Intended status: Standards Track
Expires: September 3, 2017

R. Bonica
R. Thomas
Juniper Networks
J. Linkova
Google
C. Lenart
Verizon

March 2, 2017

Extended Ping (Xping)
draft-bonica-intarea-eping-04

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 Request 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 September 3, 2017.

Table of Contents

1. Problem Statement

Network operators use Ping [RFC2151] to determine whether a remote interface is operational. 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 operational.

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:

- Global [RFC4291]
- Private [RFC1918]
- 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:

- 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.
- 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.
- 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 [RFC7404]

When a network operator renumbers 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 informational ICMP messages, called Extended Echo Request and Extended Echo Reply. The Extended Echo Request message makes a semantic distinction between the destination interface and the probed interface. The destination interface is the

interface to which the Extended Echo Request 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., both interfaces must belong to the same node).

Because the Extended Echo Request 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 Request message and Section 3 describing the Extended Echo Reply message. Section 4 describes how the probed node processes the Extended Echo Request message and Section 5 describes the Xping application. Section 6 describes uses cases.

2. ICMP Extended Echo Request

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

Figure 1 depicts the ICMP Extended Echo Request message.

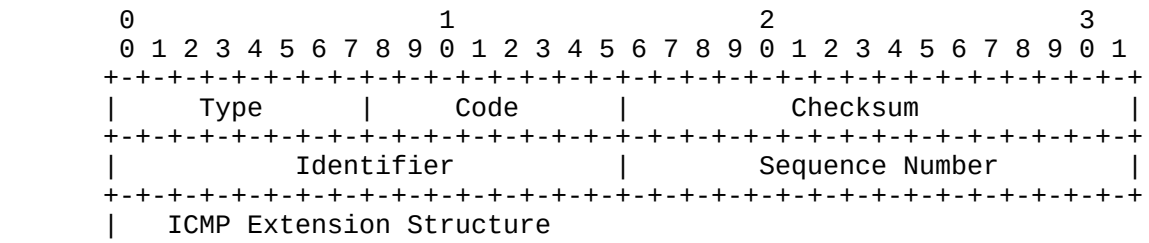


Figure 1: ICMP Extended Echo Request Message

IP Header fields:

- Source Address: The Source Address MUST be valid IPv4 or IPv6 unicast address belonging to the sending node.

- Destination Address: Identifies the destination interface (i.e., the interface to which this message will be delivered).

ICMP fields:

- Type: Extended Echo Request. The value for ICMPv4 is TBD by IANA. The value for ICMPv6 is also TBD by IANA.

- Code: 0

- Checksum: For ICMPv4, see RFC 792. For ICMPv6, see RFC 4443.

- Identifier: An identifier to aid in matching Extended Echo Replies to Extended Echo Requests. May be zero.

- Sequence Number: A sequence number to aid in matching Extended Echo Replies to Extended Echo Requests. May be zero.

- 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:

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

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

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

- An ICMPv6 Extended Echo Request 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 Request 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:

- Class-Num: Interface Identification Object. Value is TBD by IANA

- C-type: Values are: (1) Identifies Interface By Name, (2) Identifies Interface By Index, and (3) Identifies Interface By Address

- 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.

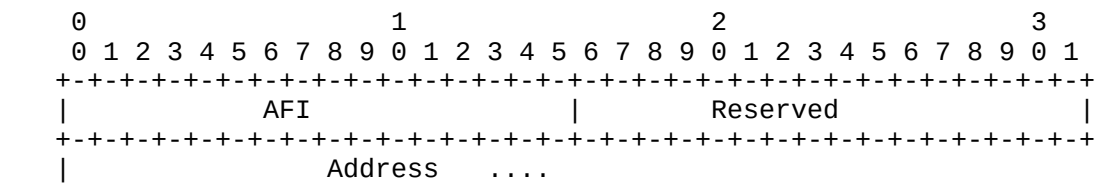


Figure 2: Interface Identification Object - C-type 3 Payload

Payload fields are defined as follows:

- 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, (6) IEEE 802, (16389) 48-bit MAC and (16390) 64-bit MAC. They MAY support other values.

- Reserved: This 16-bit field MUST be set to zero and ignored upon receipt.

- 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 defined for 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.

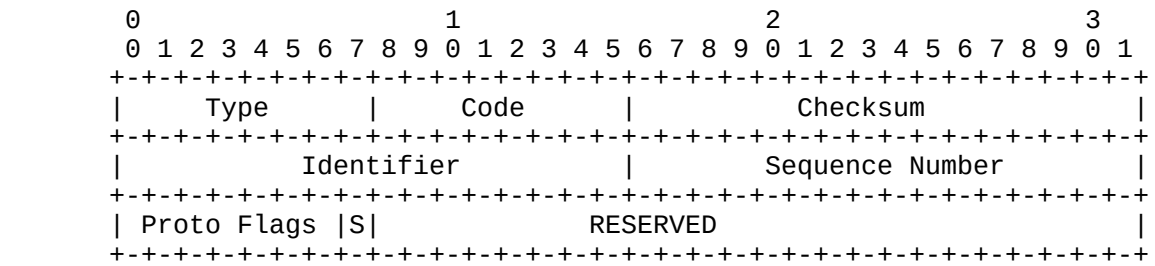


Figure 3: ICMP Extened Echo Reply Message

IP Header fields:

- Source address: Copied from the Destination Address field of the invoking Extended Echo Request message.

- Destination address: Copied from the Source Address field of the invoking Extended Echo Request message.

ICMP fields:

- Type: Extended Echo Reply. The value for ICMPv4 is TBD by IANA. The value for ICMPv6 is also TBD by IANA.

- Code: (0) No Error, (1) Malformed Query, (2) No Such Interface, (3) Multiple Interfaces Satisfy Query

- Checksum: For ICMPv4, see RFC 792. For ICMPv6, see RFC 4443.

- Identifier: Copied from the Identifier field of the invoking Extended Echo Request packet.

- Sequence Number: Copied from the Sequence Number field of the invoking Extended Echo Request packet.

- Proto 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 3-7 (Reserved)

- 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.

- Reserved: This field MUST be set to zero and ignored upon receipt.

4. ICMP Extended Echo and Extended Echo Reply Processing

When a node receives an ICMP Extended Echo Request message and any of the following conditions apply, the node MUST silently discard the incoming message:

- The node does not recognize ICMP Extended Echo Request messages
- The node has not explicitly enabled ICMP Extended Echo functionality
- The node has not explicitly enabled the incoming ICMP Extended Echo Request type (i.e., by ifName, by IfIndex, by Address)
- The incoming ICMP Extend Echo Request carries a source address that is not authorized for the incoming ICMP Extended Echo Request type
- The Source Address of the incoming messages is not a unicast address

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

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

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

- Hop Limit is 255
- Next Header is ICMPv6

In either case, the responding node MUST:

- Copy the source address from the Extended Echo Request message to the destination address of the Extended Echo Reply

- Copy the destination address from the Extended Echo Request message to the source address of the Extended Echo Reply
- Set the DiffServ codepoint to CS0 [RFC4594]
- Set the ICMP Type to Extended Echo Reply
- Copy the Identifier from the Extended Echo Request message to the Extended Echo Reply
- Copy the sequence number from the Extended Echo Request message to the Extended Echo Reply
- Set the Code field as described Section 4.1
- 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.
- If the S-bit is set, set Protocol Flags as appropriate. Otherwise, clear all Protocol Flags.
- Set the checksum appropriately
- Forward the ICMP Extended Echo Reply to its destination

The status of the probed interface is determined exactly as if it had been probed by a directly connected neighbor using traditional ping.

4.1. Code Field Processing

The following rules govern how the Code should be set:

- If the query is malformed, set the Code to Malformed Query (1)
- Otherwise, if the ICMP Extension Structure does not identify any local interfaces, set the Code to No Such Interface (2)
- Otherwise, if the ICMP Extension Structure identifies more than one local interfaces, set the Code to Multiple Interfaces Satisfy Query (3)
- Otherwise, set the code to No Error (0)

5. The Xping 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

Request, 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:

- Count
- Wait
- Source Interface Address
- Hop Count
- Destination Interface Address
- 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 Request. The Source Interface Address MUST be a unicast address and MUST identify an interface that is local to the probing node.

The destination Interface Address identifies the interface to which the ICMP Extended Echo Request message is sent. It can be an IPv4 or 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:

- an interface name

- an address from any address family (e.g., IPv4, IPv6, IEEE 802, 48-bit MAC, 64-bit MAC)
- an ifIndex

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

- an IPv6 address, whose scope is global
- an IPv6 address, whose scope is link-local
- an interface name, whose scope is node-local
- 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. Use-Cases

In the use cases below, Xping can be used to determine the operational status of a forwarding interface. Other management protocols (e.g., SNMP) might also be used to obtain this information. However, we assume that those management protocols are not viable options, either because they are too heavyweight or they are not supported on the relevant nodes.

6.1. Unnumbered Interfaces

An IPv4 network contains many routers. On each router, a loopback interface is numbered from global address space and all forwarding interfaces are unnumbered. Network operations staff need a tool that they can execute on any router in the network to determine the operational status of any forwarding interface in the network.

6.2. Link-local Interfaces

An IPv6 network contains many routers. On each router, a loopback interface is numbered from global address space and some or all forwarding interfaces are numbered from link-local address space. Network operations staff need a tool that they can execute on any router in the network to determine the operational status of any forwarding interface in the network.

6.3. Unadvertised Interfaces

A network contains many routers. On each router, the loopback interface and all forwarding interfaces are numbered from global address space. However, some forwarding interfaces do not participate in any routing protocol nor are they advertised by any routing protocol. Network operations staff need a tool that they can execute on any router in the network to determine the operational status of any forwarding interface in the network.

7. Updates to RFC 4884

Section 4.6 of RFC 4884 provides a list of extensible ICMP messages (i.e., messages that can carry the ICMP Extension Structure). This document adds the ICMP Extended Echo message and the ICMP Extended Echo Reply message to that list.

8. IANA Considerations

This document requests the following actions from IANA:

- Add an entry to the "ICMP Type Number" registry, representing the Extended Echo Request. This entry has one code (0).

- Add an entry to the "Internet Control Message Protocol version 6 (ICMPv6) Parameters" registry, representing the Extended Echo Request. This entry has one code (0).

- Add an entry to the "ICMP Type Number" registry, representing the Extended Echo Reply. This entry has the following codes: (0) No Error, (1) Malformed Query, (2) No Such Interface, (3) Multiple Interfaces Satisfy Query. Protocol Flag Bit mappings are as follows: Bit 0 (IPv4), Bit 1 (IPv6), Bit 2 (Ethernet), Bits 3-15 (Reserved).

- 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) Malformed Query, (2) No Such Interface, (3) Multiple Interfaces Satisfy Query. Protocol Flag Bit mappings are as follows: Bit 0 (IPv4), Bit 1 (IPv6), Bit 2 (Ethernet), Bits 3-15 (Reserved).

- Add an entry to the "ICMP Extension Object Classes and Class Sub-types" 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.

9. Security Considerations

The following are legitimate uses of Xping:

- to determine the operational status of an interface
- to determine which protocols (e.g., IPv4, IPv6) are active on an interface

However, malicious parties can use Xping to obtain additional information. For example, a malicious party can use Xping to discover interface names. Having discovered an interface name, the malicious party may be able to infer additional information. Additional information may include:

- interface bandwidth
- the type of device that supports the interface (e.g., vendor identity)
- the operating system version that the above-mentioned device executes

Understanding this risk, network operators establish policies that restrict access to ICMP Extended Echo functionality. In order to enforce these polices, nodes that support ICMP Extended Echo functionality MUST support the following configuration options:

- Enable/disable ICMP Extended Echo functionality. By default, ICMP Extend Echo functionality is disabled.
- Define enabled query types (i.e., by ifName, by ifIndex, by Address). By default, all query types are disabled.
- For each enabled query type, define the prefixes from which ICMP Extended Echo Request messages are permitted
- For each interface, determine whether ICMP Echo Request messages are accepted

When a node receives an ICMP Extended Echo Request message that it is not configured to support, it MUST silently discard the message. See Section 4 for details.

In order to protect local resources, implementations SHOULD rate-limit incoming ICMP Extended Echo Request messages.

10. Acknowledgements

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

11. References

11.1. 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>.

11.2. 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>.

[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>.

[RFC7404] Behringer, M. and E. Vyncke, "Using Only Link-Local Addressing inside an IPv6 Network", RFC 7404, DOI 10.17487/RFC7404, November 2014, <http://www.rfc-editor.org/info/rfc7404>.

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, Karnataka 560103
India

Email: rejithomas@juniper.net

Jen Linkova
Google
1600 Amphitheatre Parkway
Mountain View, California 94043
USA

Email: furry@google.com

Chris Lenart
Verizon
22001 Loudoun County Parkway
Ashburn, Virginia 20147
USA

Email: chris.lenart@verizon.com

intarea
Internet-Draft
Intended status: Informational
Expires: September 14, 2017

B. Bruneau
Ecole polytechnique
P. Pfister
Cisco
D. Schinazi
T. Pauly
Apple
E. Vyncke, Ed.
Cisco
March 13, 2017

Proposals to discover Provisioning Domains
draft-bruneau-intarea-provisioning-domains-00

Abstract

This document describes one possible way for hosts to retrieve additional information about their Internet access configuration. The set of configuration items required to access the Internet is called a Provisioning Domain (PvD) and is identified by a Fully Qualified Domain Name.

This document separates the way of getting the Provisioning Domain identifier, the way of getting the Provisioning Domain information and the potential information contained in the Provisioning Domain.

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 September 14, 2017.

Table of Contents

1. Introduction

It has become very common in modern networks that hosts have Internet or more specific access through different networking interfaces, tunnels, or next-hop routers. The concept of Provisioning Domain (PvD) was defined in RFC7556 [RFC7556] as a set of network configuration information which can be used by hosts in order to access the network. In this document, PvDs are associated with a Fully Qualified Domain Name (called PvD ID) which is used within the host to identify correlated sets of configuration data and also used to retrieve additional information about the services that the network provides.

Devices connected to the Internet through multiple interfaces would typically be provisioned with one PvD per interface, but it is worth noting that multiple PvDs with different PvD IDs could be provisioned on any host interface, as well as noting that the same PvD ID could be used on different interfaces in order to inform the host that both PvDs, on different interfaces, ultimately provide equivalent services.

This document proposes multiple methods allowing the host to to retrieve the PvD ID associated with a set of networking discover the PvD and retrieve the PvD information. It also explains configuration as well as the methods and format in order to retrieve some of the parameters that can describe a PvD.

2. Terminology

PvD — A provisioning domain, usually with a set of provisioning domain information; for more information, see [RFC7556].

2.1. Requirements Language

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

3. Retrieving the PvD ID

In this document, each provisioning domain is identified by a PvD ID. The PvD ID is a Fully Qualified Domain Name which belongs to the network operator to avoid conflicts among network operators. The same PvD ID can exist in several access networks if the set of configuration information is identical in all those networks (such as in all home networks of a residential subscriber). Within a host, the PvD ID SHOULD be associated to all the configuration information associated to this PvD ID; this allows for easy update and removal of information while keeping a consistent state.

This section assumes that IPv6 Router Advertisements are used to discover the PvD ID and explains why this technique was selected.

3.1. Using One Router Advertisement per PvD

Hosts receive implicit PvDs by the means of Router Advertisements (RA).

A router MAY add a single PvD ID Option in its RAs. The PvD ID specified in this option is then associated with all the Prefix Information Options (PIO) included in the RA (albeit it is expected that only one PIO will be included in the RA). All other information contained in the RA (notably the RDNSS and Route Information Option) are to be associated with the PvD ID. The set of information contained in the RA forms the bootstrap (or hint) PvD. A new RA option will be required to convey the PvD ID.

When a host receives an RA which does not include a PvD ID Option, the set of information included in the RA (such as Recursive DNS server, IPv6 prefix) is attached to an implicit PvD identified by the local interface ID on which the RA is received, and by the link-local address of the router sending the RA.

In the cases where a router should provide multiple independent PvDs to all hosts, including non-PvD aware hosts, it should send multiple RAs, as proposed in [I-D.bowbakova-rtgwg-enterprise-pa-multihoming] using different source link-local addresses (LLA); the datalink layer (MAC) address could be the same for all the different RA. If the router is actually a VRRP instance, then the procedure is identical

except that the virtual link-layer address is used as well as virtual link-layer addresses.

Using RA allows for an early discovery of the PvD ID as it is early in the interface start-up. As RA is usually processed in the kernel, this requires a host OS upgrade. The RA SHOULD contain other PvD information as explained in section Section 4.1.

3.2. Rationale for not selecting other techniques

There are other techniques to discover the PvD ID that were not selected by the authors and reviewers, this section explains why. The design goal was to be as reliable as possible (do not depend on Internet connectivity) and as fast as possible.

3.2.1. Using DNS-SD

For each received RA including a RDNSS option as well as a DNS search list option, the host MAY retrieve the PvD ID by querying the configured DNS server for records of type PTR associated with _pvd.<DNS search name>. If a PvD ID is configured, the DNS recursive resolver MUST reply with the PvD ID as a PTR record. NXDOMAIN is returned otherwise.

When the RDNSS address is link-local, the host MAY retrieve the PvD ID before configuring its global scope address(es).

Relying on a valid DNS service at the interface bootstrap can lead into delay to start the interface or starting without enough information: for example when the RDNSS is a non local address and there is no Internet connectivity.

3.2.2. Using Reverse DNS lookup

[I-D.stenberg-mif-mpvd-dns] proposes a solution to get the name of the PvD using a reverse DNS lookup based on the host global address(es). It merely relies on prepending a well-known prefix '_pvd' to the reverse lookup, for example ' _pvd....ip6.arpa.'.

However, the PvD information is typically provided by the network operator, whereas the reverse DNS zone could be delegated from the operator to the network user, in which case it would not work.

It also requires a fully functional global address to retrieve the information which may be too late for a correct host configuration. One advantage is that it does not require any change in the IPv6 protocol and no change in the host kernel or even in the CPE.

3.3. IoT Considerations

TBD: should state that when end-host (IoT) cannot impletement completely this RFC it MAY select any of the PvD or the router SHOULD send a single unicast RA (hence a single PvD) in response to the RS or none if it detects that it cannot offer the right set of network services.

3.4. Linking IPv4 Information to an IPv6 PvD

The document describes IPv6-only PvD but there are multiple ways to link the set of IPv4 configuration information received by DHCPv4:

- correlation based on the data-link layer address of the source, if the IPv6 RA and the DHCPv4 response have the same data-link layer address, then the information contained in the IPv4 DHCP can be linked to the IPv6 PvD;

- correlation based on the interface when there is no data-link address on the link (such as a 3GPP link), then the information contained in the IPv4 PDP context can be linked to the IPv6 PvD (*** TO BE VERIFIED before going -01);

- correlation based on the DNS search list, if the DNS search lists are identical between the IPv6 RDNSS and the DHCPV4 response, then the information contained in the IPv4 DHCP response can be linked to the IPv6 PvD.

The correlation could be useful for some PvD information such as Internet reachability, use of captive portal, display name of the PvD, ...

In cases where the IPv4 configuration information could not be associated with a PvD, hosts MUST consider it as attached to an independent implicit PvD containing no other information than what is provided through DHCPv4.

4. Getting the full set of PvD information

Once the PvD ID is known, it MAY be used to retrieve additional information. PvD Information is modeled as a key-value dictionary which keys are ASCII strings of arbitrary length, and values are either strings (encoding can vary), ordered list of values (recursively), or a dictionary (recursively).

The PvD Information may be retrieved from multiple sources (from the bootstrap PvD contained in the RA to the secondary/extended PvD described in this section); the PvD ID is then used to correlate the

information from different sources. The way a host should operate when receiving conflicting information is TBD but it SHOULD at least override information from less authenticated sources (RA) by more authenticated sources (via TLS).

4.1. Using the PvD Bootstrap Information Option

Routers MAY transmit, in addition to the PvD ID option, a PvD Bootstrap Information option, containing a first subset of PvD information. The additional pieces of bootstrap PvD information data set are transmitted using the short-hand notation proposed in Section 5. This requires another RA option.

As there is a size limit on the amount of information a single RA can convey, it is likely that the PvD Bootstrap Information option may not contain the whole set of PvD Information. The set of PvD information included in the RA is called PvD Bootstrap Information.

4.2. Downloading a JSON file over HTTPS

The host SHOULD try to download a JSON formatted file over HTTPS in order to get more PvD information.

The host MUST perform an HTTP query to https://<PvD-ID>/v1.json. If the HTTP status of the answer is greater than 400 the host MUST abandon and consider that there is no additional PvD information. If the HTTP status of the answer is between 300 and 400 it MUST follow the redirection(s). If the HTTP status of the answer is between 200 and 300 the host MAY get a file containing a single JSON object.

The host MUST respect the cache information in the HTTP header, if any, and at expiration of the downloaded object, it must fetch a fresher version if any.

4.2.1. Advantages

The JSON format allows advanced structures.

It can be secured using HTTPS (and DNSSEC).

It is easier to update a file on a web server than to edit DNS records. It can be especially important if we want providers to be able to often update the remaining phone plan of the user.

4.2.2. Disadvantages

It is slower than using DNS because HTTPS uses TCP and TLS and needs more packets to be exchanged to get the file.

An additional HTTPS server must be deployed and configured.

4.3. Using DNS TXT ressource records (not selected)

This approach was not selected during the design team meeting but has kept here for reference, it will be removed after global consensus is reached.

The host could perform a DNS query for TXT resource records (RR) for the FQDN used as PvD ID (alternatively for _pvd.<PvD-ID>). For each retrieved PvD ID, the DNS query MUST be sent to the DNS server configured from the same router advertisement as the PvD ID. Syntax of the TXT response is defined in Section 5 (Section 5).

4.3.1. Advantages

It requires a single round-time trip in order to retrieve the PvD Information.

It can be secured using DNSSEC.

4.3.2. Disadvantages

A TXT record is limited to 65535 characters in theory but large size of TXT records could require either DNS over TCP (so loosing the 1-RTT advantage) or fragmented UDP packets (which could be dropped by a bad choice of security policy). Large TXT records could also be used to mount an amplification attack.

4.3.3. Using DNS SRV ressource records

It is expected that the DNS TXT records will be sufficient for the host to configure itself with basic networking and policy configuration. Nevertheless, if further information is required, or when a different security model shall be used to access the PvD Information, a SRV Resource Record including a full URL MAY be included as a response, expecting the host to query this URL in order to retrieve additional PvD information.

5. PvD Information

PvD information is a set of key-value pairs. Keys are ASCII character strings. Values are either a character string, an ordered list of values, or an embedded dictionary. Value types and default behavior with respect to some specific keys MAY be further specified (recursively). Some keys have a default value as described in the following sections. When there is an expiration time in a PvD, then the information MUST be refreshed before the expiration time. The behavior of a host when the refresh operation is not successful is TBD.

Nodes using the PvD MUST support the two encodings:

JSON syntax for the complete set of PvD information;

short-hand notation for the bootstrap PvD.

When the PvD information is transferred as a JSON file, then the key used is the second column of the following table. The syntax of the JSON file is obvioulsy JSON and is richer than the short-hand notation specified in the next paragraph.

When transmitting more information than the PvD ID in the RA (or when DNS TXT resource records are used), the shorthand notataion for PvD information is used and consists of a string containing several "key=value;" substrings. The "key" is the first column of the following tables, the value is encoded as:

Shorthand notation for values:

integer: expressed in decimal format with a '.' (dot) used for decimals;

string: expressed as UTF-8 encoded string, delimited by single quote character, the single quote character can be expressed by two consecutive single quote character;

boolean: expressed as '0' for false and '1' for true;

IPv6 address: printed as RFC5952 [RFC5952].

5.1. PvD Name

PvD SHOULD have a human readable name in order to be presented on a GUI. The name can also be localized.

DNS TXT key/Bootstrap PvD key	JSON key	Description	Type	JSON Example
n	name	User-visible service name, SHOULD be part of the bootstrap PvD	human-readable UTF-8 string	"Foobar Service"
nl10n	localizedName	Localized user-visible service name, language can be selected based on the HTTP Accept-Language header in the request.	human-readable UTF-8 string	"Service Blabla"

5.2. Trust of the bootstrap PvD

The content of the bootstrap PvD (from the original RA) cannot be trusted as it is not authenticated. But, the extended PvD can be associated with the PvD ID (as the PvD ID is used to construct the extended PvD URL) and trusted by the used of TLS. The extended PvD SHOULD therefore include the following information elements and, if they are present, the host MUST verify that the all PIO of the RA fits into the master prefix list. If any PIO prefix from the bootstrap PvD does not fit in the master prefix array, then all information received by the bootstrap PvD must be invalidated. In short, the masterIPv6Prefix received over TLS is used to authenticate the bootstrap PvD.

The values of the bootstrap PvD (RDNSS, ...) are overwritten by the values contained in the trusted extended PvD if they are present.

DNS TXT key	JSON key	Description	Type	JSON Example
mp6	masterIpv6Prefix	All the IPv6 prefixes linked to this PvD (such as a /29 for the ISP).	Array of IPv6 prefixes	["2001:db8::/32 "]

5.3. Reachability

The following set of keys can be used to specify the set of services for which the respective PvD should be used. If present they MUST be honored by the client, i.e., if the PvD is marked as not usable for Internet access (walled garden), then it MUST NOT be used for Internet access. If the usability is limited to a certain set of domain or address prefixes (typical VPN access), then a different PvD MUST be used for other destinations.

DNS TXT key	JSON key	Description	Type	JSON Example
s	noInternet	Internet inaccessible	boolean	true
cp	captivePortal	Presence of a captive portal	boolean	false
z	dnsZones	DNS zones accessible and searchable	array of DNS zone	["foo.com","sub.bar.com"]
6	prefixes6	IPv6-prefixes accessible via this PvD	array of IPv6 prefixes	["2001:db8:a::/48","2001:db8:b:c::/64"]
4	prefixes4	IPv4-prefixes accessible	array of IPv4 prefixes in CIDR reachable via this PvD	["192.0.2.0/24","2.3.0.0/16"]

5.4. DNS Configuration

The following set of keys can be used to specify the DNS configuration for the respective PvD. If present, they MUST be honored and used by the client whenever it wishes to access a resource described by the PvD.

DNS TXT key	JSON key	Description	Value	JSON Example
r	dnsServers	Recursive DNS server	array of IPv6 and IPv4 addresses	["2001:db8::1","192.0.2.2"]
d	dnsSearch	DNS search domains	array of search domains	["foo.com","sub.bar.com"]

5.5. Connectivity Characteristics

NOTE: open question to the authors/reviewers: should this document include this section or is it useless?

The following set of keys can be used to signal certain characteristics of the connection towards the PvD.

They should reflect characteristics of the overall access technology which is not limited to the link the host is connected to, but rather a combination of the link technology, CPE upstream connectivity, and further quality of service considerations.

DNS TXT key	JSON key	Description	Type	JSON Example
tp	throughputMax	Maximum achievable throughput (e.g. CPE downlink/uplink)	object({down (int), up(int)}) in kb/s	{"down": 10000, "up": 5000}
lt	latencyMin	Minimum achievable latency	object({down (int), up(int)}) in ms	{"down": 10, "up": 20}
rl	reliabilityMax	Maximum achievable reliability	object({down (int), up(int)}) in 1/1000	{"down": 1000, "up": 800}
cp	captivePortal	Captive portal	URL of the portal	"https://example.com"
nat	NAT	IPv4 NAT in place	boolean	true
natto	NAT Time-out	The value in seconds of the NAT time-out	Integer	30
srh	segmentRoutingHeader	The IPv6 Segment Routing Header to be used between the IPv6 header and	Binary string	...

		any other headers when using this PvD		
srhDNS	segmentRoutingHeaderDnsFQDN	The DNS FQDN which is used to retrieved the actual IPv6 Segment Routing Header to be used between the IPv6 header and any other headers when using this PvD	Ascii string	srh.pvd-foo.example.org
cost	cost	Cost of using the connection	object	See Section 5.6

5.6. Connection monetary cost

NOTE: This section is included as a request for comment on the potential use and syntax.

The billing of a connection can be done in a lot of different ways. The user can have a global traffic threshold per month, after which his throughput is limited, or after which he/she pays each megabyte. He/she can also have an unlimited access to some websites, or an unlimited access during the weekends.

We propose to split the final billing in elementary billings, which have conditions (a start date, an end date, a destination IP address...). The global billing is an ordered list of elementary billings. To know the cost of a transmission, the host goes through the list, and the first elementary billing whose the conditions are fulfilled gives the cost. If no elementary billing conditions match the request, the host MUST make no assumption about the cost.

5.6.1. Conditions

Here are the potential conditions for an elementary billing. All conditions MUST be fulfill.

Note: the final version should use short-hand key names.

Key	Description	Type	JSON Example
beginDate	Date before which the billing is not valid	ISO 8601	"1977-04-22T06:00:00Z"
endDate	Date after which the billing is not valid	ISO 8601	"1977-04-22T06:00:00Z"
domains	FQDNs whose the billing is limited	array(string)	["deezer.com","spotify.com"]
prefixes4	IPv4 prefixes whose the billing is limited	array(string)	["78.40.123.182/32","78.40.123.183/32"]
prefixes6	IPv6 prefixes whose the billing is limited	array(string)	["2a00:1450:4007:80e::200e/64"]

5.6.2. Price

Here are the different possibilities for the cost of an elementary billing. A missing key means "all/unlimited/unrestricted". If the elementary billing selected has a trafficRemaining of 0 kb, then it means that the user has no access to the network. Actually, if the last elementary billing has a trafficRemaining parameter, it means that when the user will reach the threshold, he/she will not have access to the network anymore.

Key	Description	Type	JSON Example
pricePerGb	The price per Gigabit	float (currency per Gb)	2
currency	The currency used	ISO 4217	"EUR"
throughputMax	The maximum achievable throughput	float (kb/s)	1000
trafficRemaining	The traffic remaining	float (kb)	96000000

5.6.3. Examples

Example for a user with 20 GB per month for 40 EUR, then reach a threshold, and with unlimited data during weekends and to deezer:

```
[
  {
    "domains": ["deezer.com"]
  },
  {
    "prefixes4": ["78.40.123.182/32","78.40.123.183/32"]
  },
  {
    "beginDate": "2016-07-16T00:00:00Z",
    "endDate": "2016-07-17T23:59:59Z",
  },
  {
    "beginDate": "2016-06-20T00:00:00Z",
    "endDate": "2016-07-19T23:59:59Z",
    "trafficRemaining": 96000000
  },
  {
    "throughputMax": 1000
  }
]
```

If the host tries to download data from deezer.com, the conditions of the first elementary billing are fulfilled, so the host takes this elementary billing, finds no cost indication in it and so deduces that it is totally free. If the host tries to exchange data with youtube.com and the date is 2016-07-14T19:00:00Z, the conditions of the first, second and third elementary billing are not fulfilled. But the conditions of the fourth are. So the host takes this

elementary billing and sees that there is a threshold, 12 GB are remaining.

Another example for a user abroad, who has 3 GB per year abroad, and then pay each MB:

```
[
  {
    "beginDate": "2016-02-10T00:00:00Z",
    "endDate": "2017-02-09T23:59:59Z",
    "trafficRemaining": 9200000
  },
  {
    "pricePerGb": 30,
    "currency": "EUR"
  }
]
```

5.7. Private Extensions

keys starting with "x-" are reserved for private use and can be utilized to provide vendor-, user- or enterprise-specific information. It is RECOMMENDED to use one of the patterns "x-FQDN-KEY" or "x-PEN-KEY" where FQDN is a fully qualified domain name or PEN is a private enterprise number [PEN] under control of the author of the extension to avoid collisions.

5.8. Examples

5.8.1. Using JSON

```
{
  "name": "Orange France",
  "localizedName": "Orange France",
  "dnsServers": ["8.8.8.8", "8.8.4.4"],
  "throughputMax": {
    "down": 100000,
    "up": 20000
  },
  "cost": [
    {
      "domains": ["deezer.com"]
    },
    {
      "prefixes4": ["78.40.123.182/32","78.40.123.183/32"]
    },
    {
      "beginDate": "2016-07-16T00:00:00Z",
      "endDate": "2016-07-17T23:59:59Z",
    },
    {
      "beginDate": "2016-06-20T00:00:00Z",
      "endDate": "2016-07-19T23:59:59Z",
      "trafficRemaining": 96000000
    },
    {
      "throughputMax": 1000
    }
  ]
}
```

5.8.2. Using DNS TXT records

```
n=Orange France
r=8.8.8.8,8.8.4.4
tp=100000,20000
cost+0+domains=deezer.com
cost+1+prefixes4=78.40.123.182/32,78.40.123.183/32
cost+2+beginDate=2016-07-16T00:00:00Z
cost+2+endDate=2016-07-17T23:59:59Z
cost+3+beginDate=2016-06-20T00:00:00Z
cost+3+endDate=2016-07-19T23:59:59Z
cost+3+trafficRemaining=96000000
cost+4+throughputMax=1000
```

6. Use case examples

TBD: 1 or 2 examples when PvD are critical

6.1. Multihoming

First example could be multihoming (very much in-line with bowbakova draft).

6.2. VPN/Extranet example

using PvD to reach a specific destination (such as VPN or extranet).

7. Security Considerations

While the PvD ID can be forged easily, if the host retrieve the extended PvD via TLS, then the host can trust the content of the extended PvD and verifies that the RA prefix(es) are indeed included in the master prefixed of the extended PvD.

8. Acknowledgements

Many thanks to M. Stenberg and S. Barth: Section 5.3, Section 5.5 and Section 5.7 are from their document [I-D.stenberg-mif-mpvd-dns].

Thanks also to Ray Bellis, Lorenzo Colitti, Marcus Keane, Erik Kline, Jen Lenkova, Mark Townsley and James Woodyatt for useful and interesting brainstorming sessions.

9. References

9.1. 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>.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <http://www.rfc-editor.org/info/rfc5952>.

[RFC7556] Anipko, D., Ed., "Multiple Provisioning Domain Architecture", RFC 7556, DOI 10.17487/RFC7556, June 2015, <http://www.rfc-editor.org/info/rfc7556>.

9.2. Informative references

[I-D.bowbakova-rtgwg-enterprise-pa-multihoming]
Baker, F., Bowers, C., and J. Linkova, "Enterprise Multihoming using Provider-Assigned Addresses without Network Prefix Translation: Requirements and Solution", draft-bowbakova-rtgwg-enterprise-pa-multihoming-01 (work in progress), October 2016.

[I-D.stenberg-mif-mpvd-dns]
Stenberg, M. and S. Barth, "Multiple Provisioning Domains using Domain Name System", draft-stenberg-mif-mpvd-dns-00 (work in progress), October 2015.

[PEN] IANA, "Private Enterprise Numbers", <https://www.iana.org/assignments/enterprise-numbers>.

Authors' Addresses

Basile Bruneau
Ecole polytechnique
Vannes 56000
France

Email: basile.bruneau@polytechnique.edu

Pierre Pfister
Cisco
11 Rue Camille Desmoulins
Issy-les-Moulineaux 92130
France

Email: ppfister@cisco.com

David Schinazi
Apple

Email: dschinazi@apple.com

Tommy Pauly
Apple

Email: tpauly@apple.com

Eric Vyncke (editor)
Cisco
De Kleetlaan, 6
Diegem 1831
Belgium

Email: evyncke@cisco.com

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/1id-abstracts.html

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

Copyright and License Notice

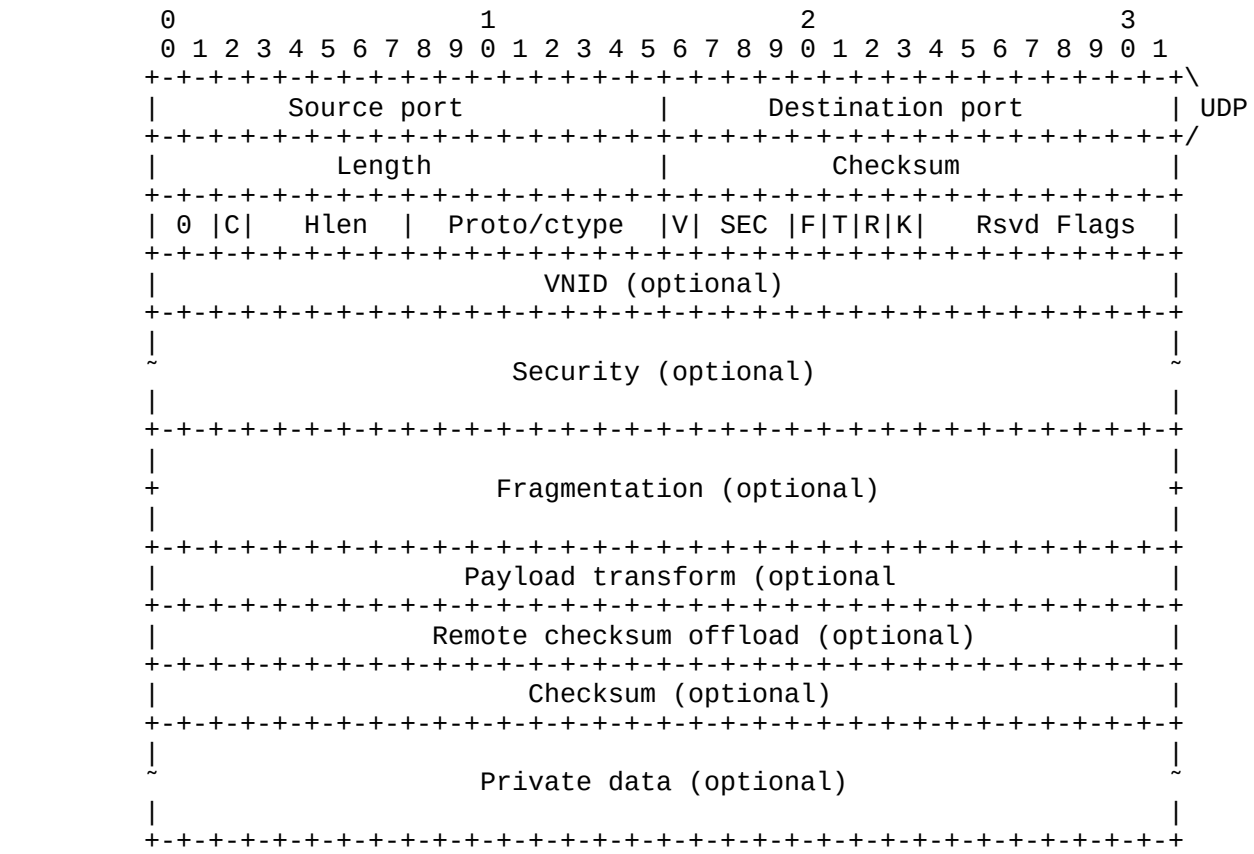
Table of Contents

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:

- Ver: Version. Set to 0 to indicate GUE encapsulation header. Note that version 1 does not allow options.
- 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.
- 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 (header_len - 4) / 4. The length of the encapsulated packet is determined from the UDP length and the Hlen: encapsulated_packet_length = UDP_Length - 12 - 4*Hlen.
- 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.
- V: Indicates the network virtualization extension (VNID) field is present. The VNID option is described in [I.D.hy-nvo3-gue-4-nvo].
- SEC: Indicates security extension field is present. The security option is described in section 3.
- F: Indicates fragmentation extension field is present. The fragmentation option is described in section 4.
- T: Indicates payload transform extension field is present. The payload transform option is described in section 5.
- R: Indicates the remote checksum extension field is present. The remote checksum offload option is described in section 6.
- K: Indicates checksum extension field is present. The checksum option is described in section 7.
- Private data is described in [I.D.nvo3-gue].

3. Security option

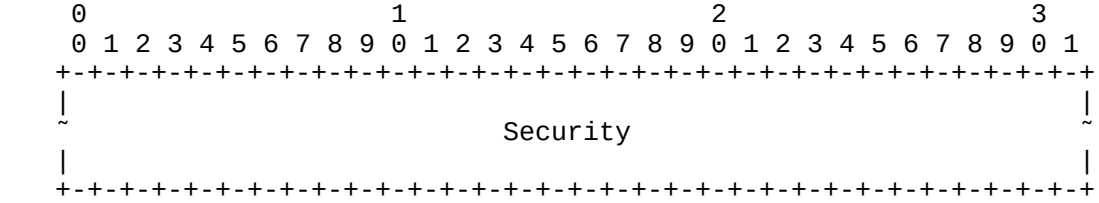
The GUE security option provides origin authentication and integrity

protection of the GUE header at tunnel end points to guarantee isolation between tunnels and mitigate Denial of Service attacks.

3.1. Extension field format

The presence of the GUE security option is indicated in the SEC flag bits of the GUE header.

The format of the security option is:



The fields of the option are:

- Security (variable length). Contains the security information. The specific semantics and format of this field is expected to be negotiated between the two communicating nodes.

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:

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

3.2. Usage

The GUE security field should be used to provide integrity and authentication of the GUE header. Security parameters (interpretation of security field, key management, etc.) are expected to be

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 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.

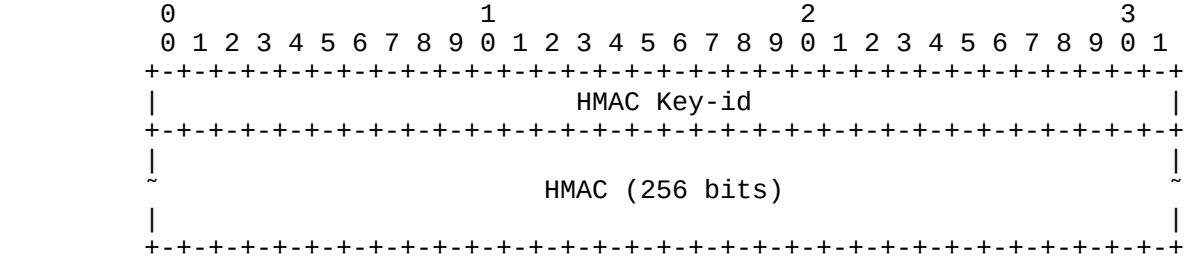
3.4. HMAC

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:



Fields are:

- HMAC Key-id: opaque field to allow multiple hash algorithms or key selection
- 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:

- The IP addresses
- 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:

- 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.
- 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:

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

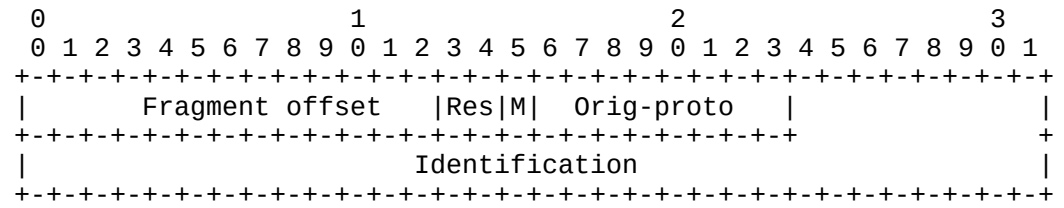
4.2. Scope

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:



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:

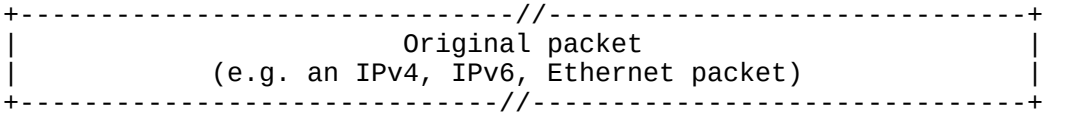
- C-bit: This is set for each fragment based on the whether the original packet being fragmented is a control or data message.
- Proto/ctype - For the first fragment (fragment offset is zero) this is set to that of the original packet being fragmented (either will be a control type or IP protocol). For 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.
- 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).

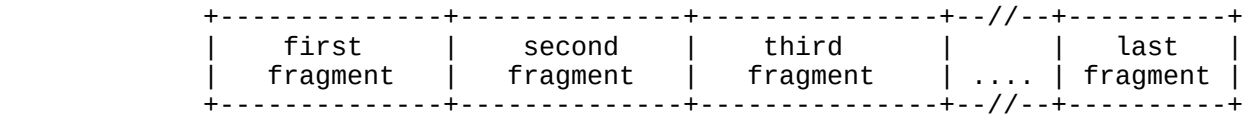
For every packet that is to be fragmented, the source node generates an Identification value. The Identification must be different than that of any other fragmented packet sent within the past 60 seconds (Maximum Segment Lifetime) with the same tunnel identification-- that is the same outer source and 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:

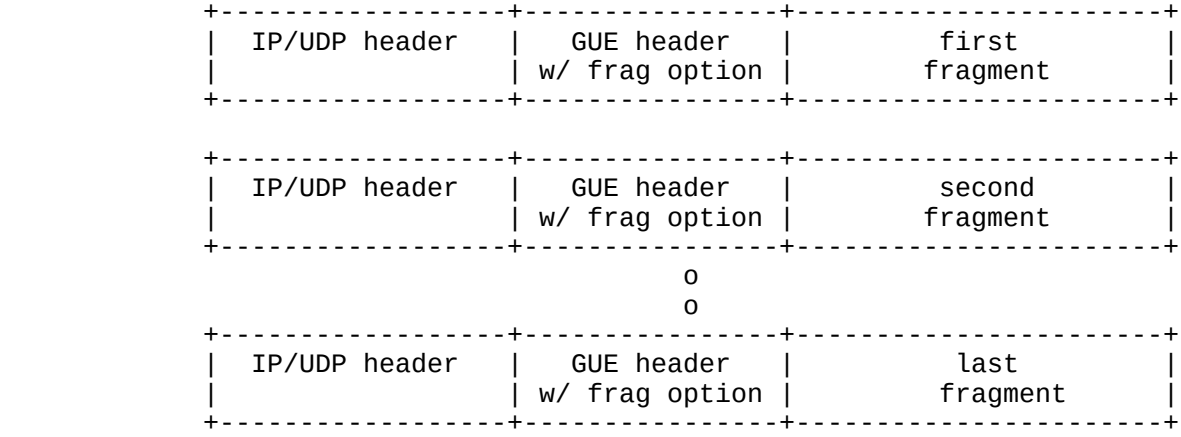


Fragmentation and encapsulation are performed on the original packet in sequence. First the packet is divided up in to fragments, and then each fragment is encapsulated. Each fragment, except possibly the last ("rightmost") one, is an integer multiple of 8 octets long. Fragments MUST be non-overlapping.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

- The IP addresses and UDP ports must be the same for all fragments of a fragmented packet.

(2) A GUE header that contains:

- 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.

- 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.

- 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.

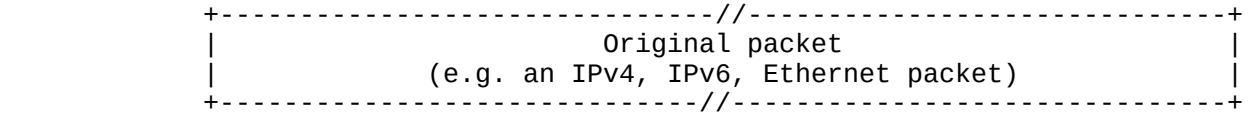
(3) The GUE fragmentation option. The contents of the extension field include:

- 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:



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:

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

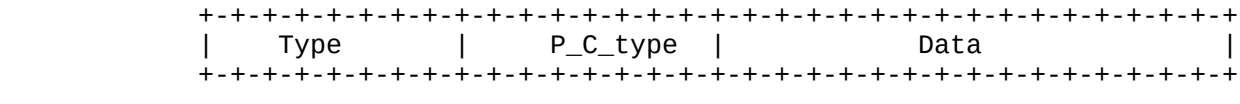
5. Payload transform option

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:



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 decaspsulator 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.

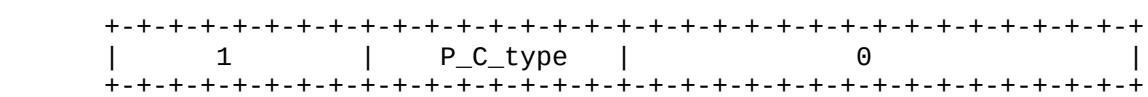
If both the security option and the payload transform are used in a GUE packet, an encapsulator must perform the payload transformation first, set the payload transform option in the GUE header, and then create the security option. A decapsulator does processing in reverse-- the security option is processed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



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

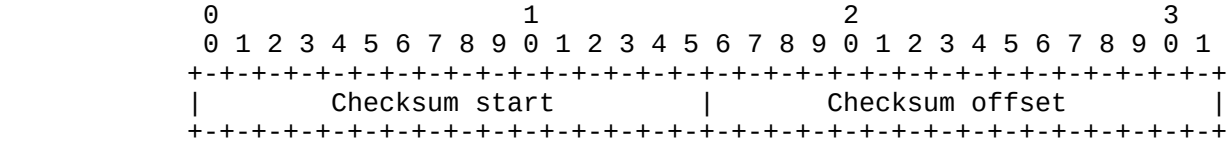
implementations can only offload the outer UDP checksum in UDP encapsulation. Remote checksum offload is described in [UDPENCAP].

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:



The fields of the option are:

- 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).

- 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 then 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.

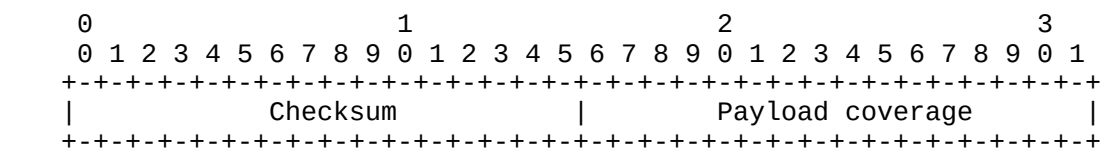
7. Checksum option

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:

- 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).

- 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.

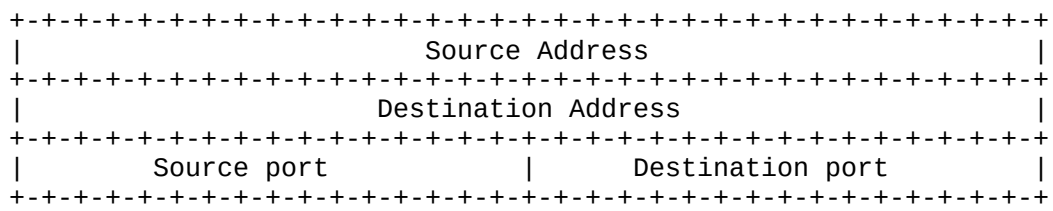
The GUE header checksum should be used when the UDP checksum is zero for IPv4 if the GUE header includes data that when corrupted can lead to misdelivery or other serious consequences, and there is no other mechanism that provides protection (no security field that checks integrity for instance).

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).

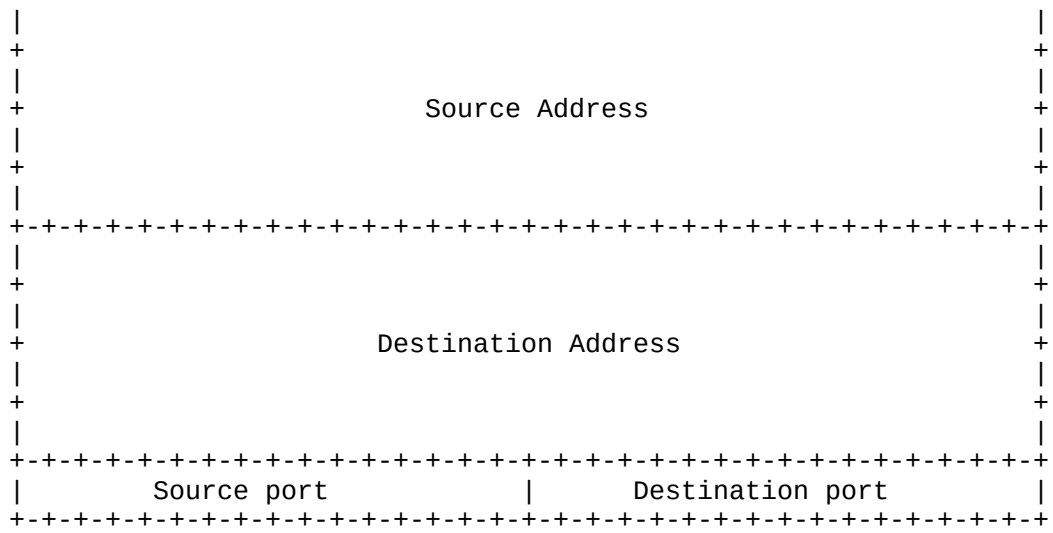
7.3. GUE checksum pseudo header

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:



The GUE pseudo header for IPv6 is:



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:

- If the length is corrupted this will usually be detected by a checksum validation failure on the inner packet.
- 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.
- A corrupted length field in itself should not lead to misdelivery of a packet.

- 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: September 14, 2017

Tom Herbert
Quantonium
Petr Lapukhov
Facebook
March 13, 2017

Identifier-locator addressing for IPv6
draft-herbert-nvo3-ila-04

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.

Table of Contents

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.

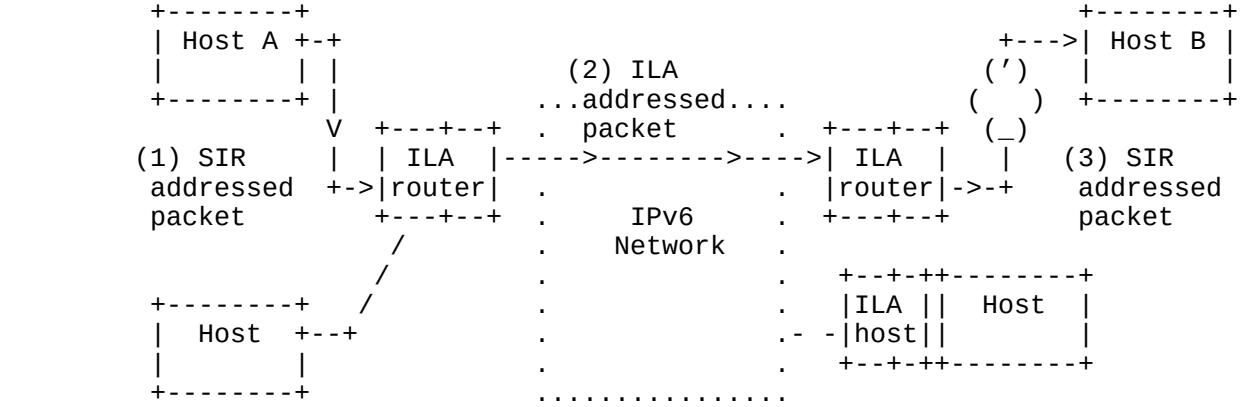
ILA translations can transform addresses from one type to another. In network virtualization virtual addresses can be translated into ILA and SIR addresses, and conversely ILA and SIR addresses can be translated to virtual addresses.

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

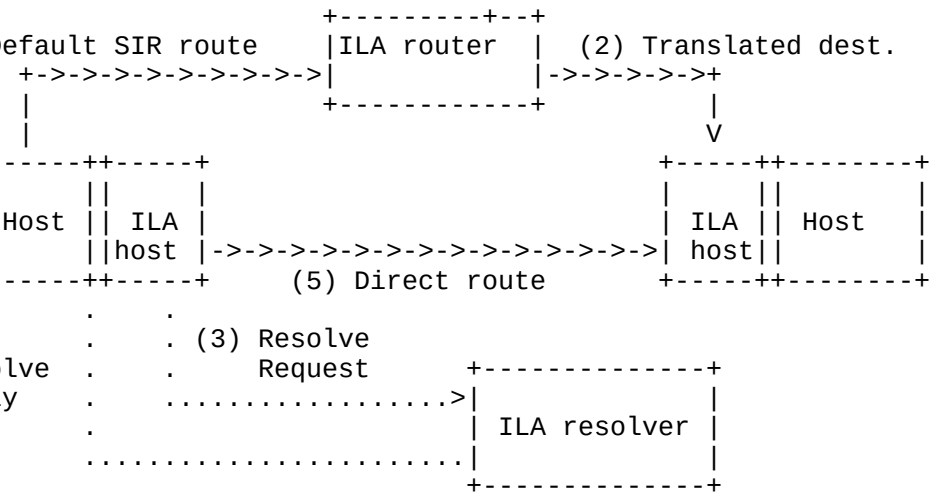
Address translation is the mechanism employed by ILA. Logical or virtual addresses are translated to topological IPv6 addresses for transport to the proper destination. Translation occurs in the upper sixty-four bits of an address, the low order sixty-four bits contains an identifier that is immutable 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)).

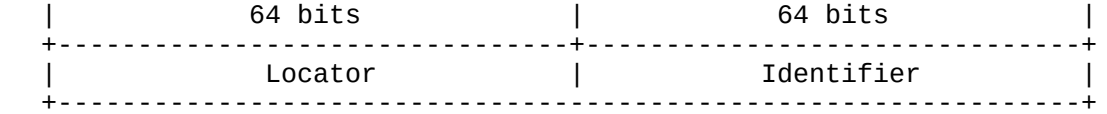
If a sending host is ILA capable the triangular routing can be eliminated by performing an ILA resolution protocol. This entails the host sending an ILA resolve request that specifies the SIR address to resolve (step (3) in the figure). An ILA resolver can respond to a resolver request with the identifier 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

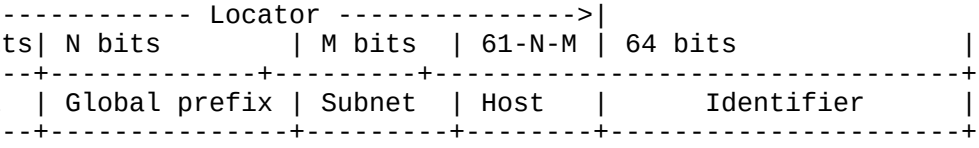
An ILA address is composed of a locator and an identifier where each occupies sixty-four bits (similar to the encoding in ILNP [RFC6741]).



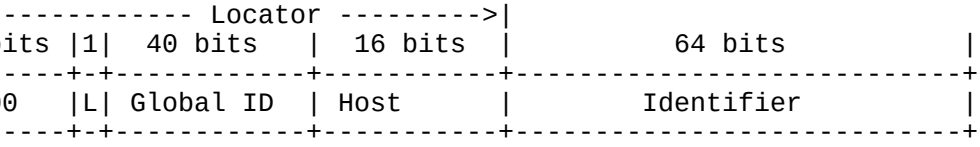
3.2 Locators

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:

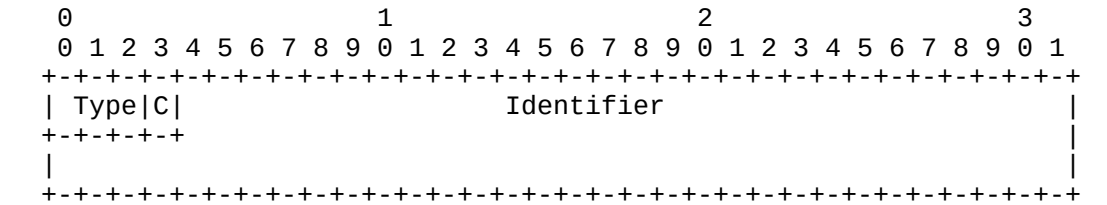


The format of an ILA address with a unique local IPv6 unicast locator is:



3.3 Identifiers

The format of an ILA identifier is:

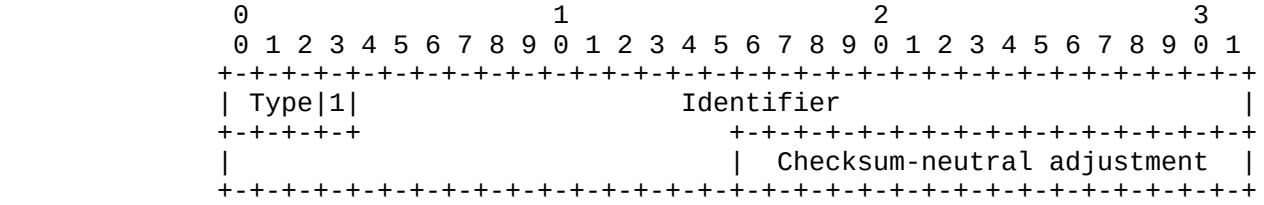


Fields are:

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

3.3.1 Checksum neutral-mapping format

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:



3.3.2 Identifier types

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

0: interface identifier

1: locally unique identifier

2: virtual networking identifier for IPv4 address

3: virtual networking identifier for IPv6 unicast address

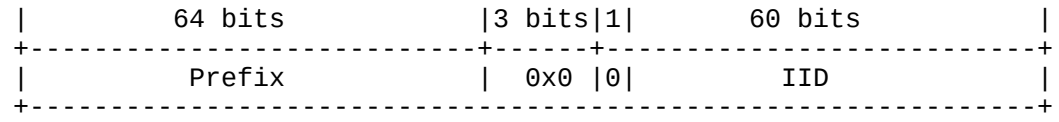
4: virtual networking identifier for IPv6 multicast address

5-7: Reserved

3.3.2.1 Interface identifiers

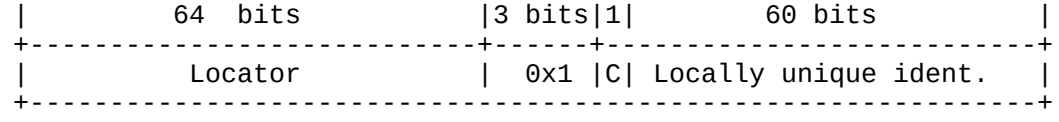
The interface identifier type indicates a plain local scope interface identifier. When this type is used the address is a normal IPv6 address without identifier-locator semantics. The purpose of this type is to allow normal IPv6 addresses to be defined within the same networking prefix as ILA addresses. Type bits and C-bit MUST be zero.

The format of an ILA interface identifier address is:

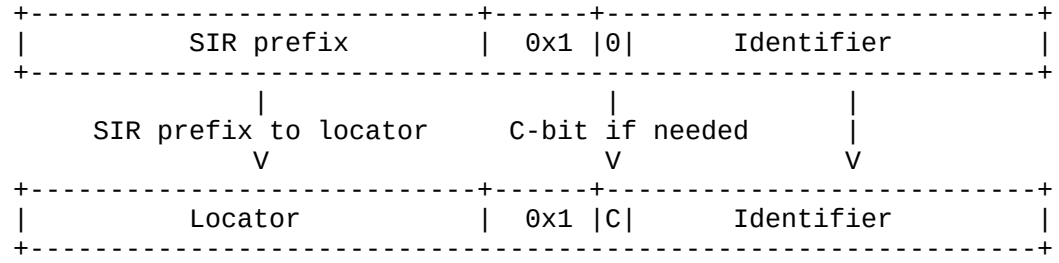


3.3.2.2 Locally unique identifiers

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:

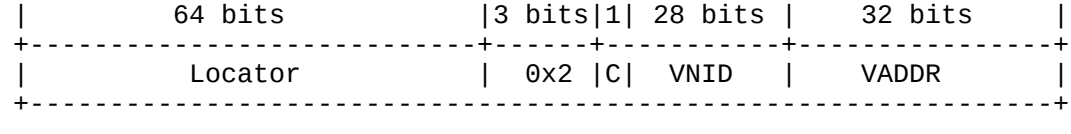


The figure below illustrates the translation from SIR address to an ILA address as would be performed when a node sends to a SIR address. Note the low order 16 bites of the identifier may be modified as the checksum-neutral adjustment. The reverse translation of ILA address to SIR address is symmetric.



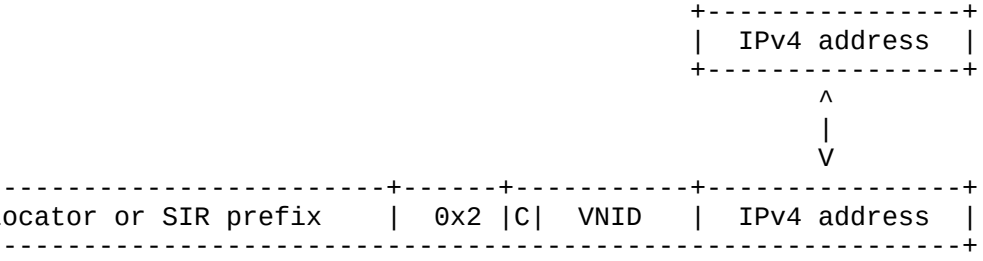
3.3.2.3 Virtual networking identifiers for IPv4

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:



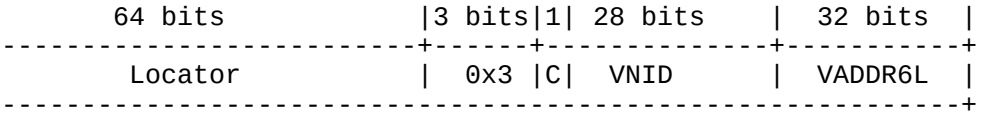
VNID is a virtual network identifier and VADDR is a virtual address within the virtual network indicated by the VNID. The VADDR can be an IPv4 unicast or multicast address, and may often be in a private address space (i.e. [RFC1918]) used in the virtual network.

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.



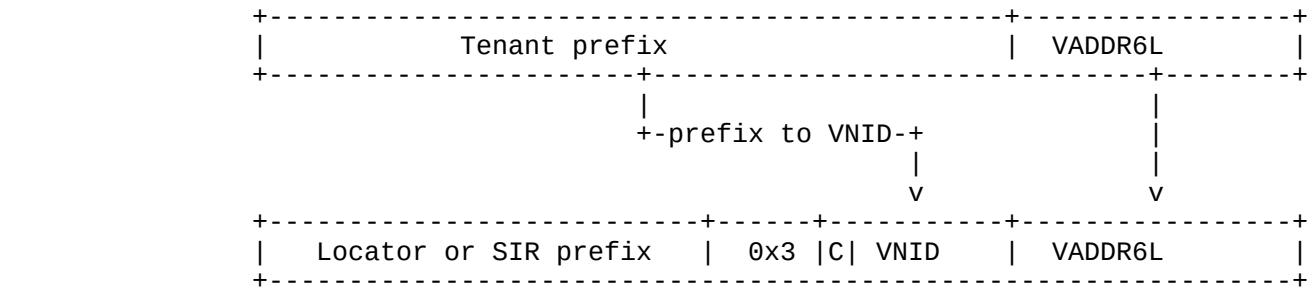
3.3.2.4 Virtual networking identifiers for IPv6 unicast

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:

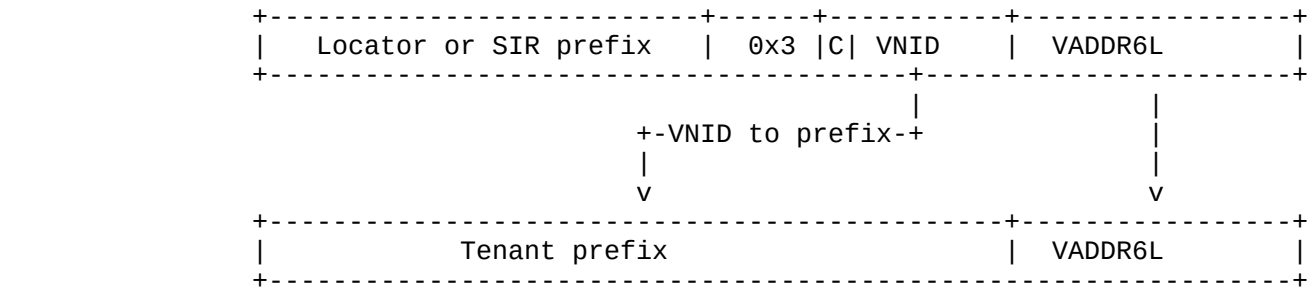


VADDR6L contains the low order 32 bits of the IPv6 virtual address. The upper 96 bits of the virtual address are inferred from the VNID to prefix mapping. Note that for ILA translations the low order sixteen of the VADDR6L may be modified for checksum-neutral adjustment.

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

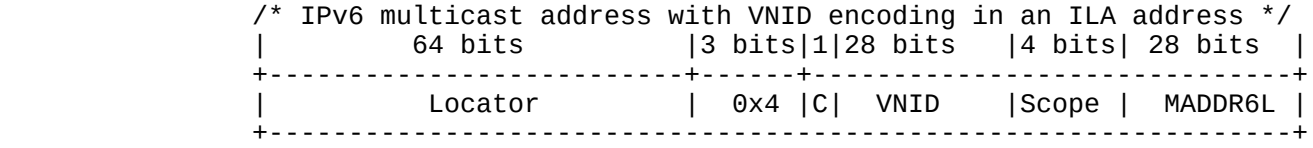


This encoding is reversible, given an ILA address, the virtual address visible to the tenant can be deduced:



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:

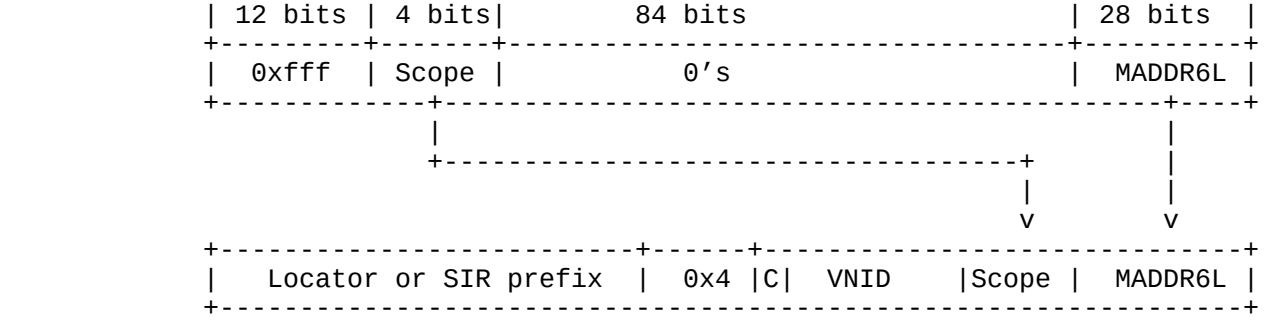
ff0<Scope>::<MADDRL6 high 12 bits>:<MADDRL6 low 16 bits>

And so can encode multicast addresses of the form:

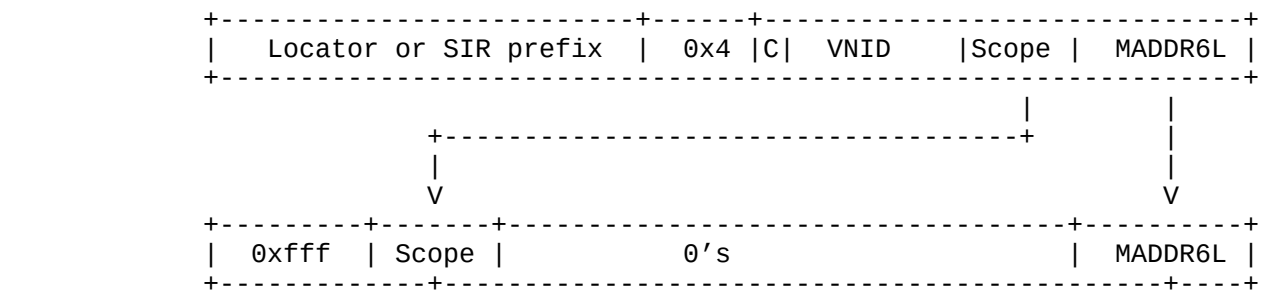
ff0X::0 to ff0X::0fff:ffff

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.



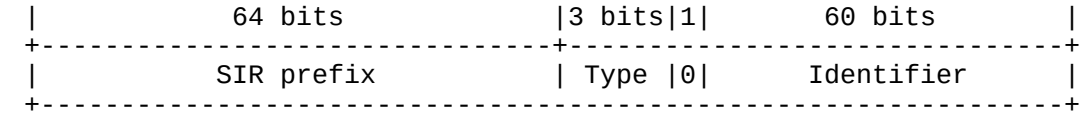
This translation is reversible:



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:



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

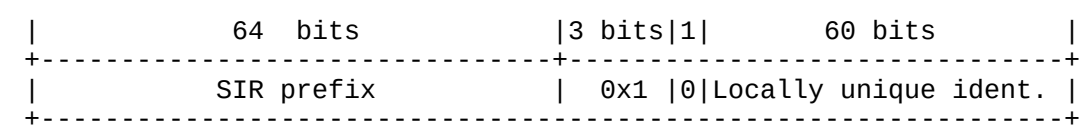
A SIR prefix may be site-local, or globally routable. A globally routable SIR prefix facilitates connectivity between hosts on the Internet and ILA nodes. A gateway between a site's network and the Internet can translate between SIR prefix and locator for an identifier. A network may have multiple SIR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

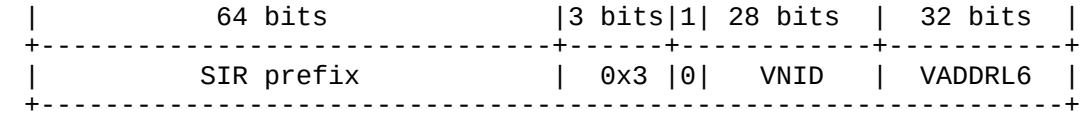
3.4.1 SIR for locally unique identifiers

The SIR address for a locally unique identifier has format:



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:



Note that this allows three representations of the same address in he 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 do deal with this:

- Perform a checksum-neutral mapping to ensure that an encapsulated transport layer checksum is kept correct on the wire.

- 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. 0xefff) 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 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 for applications. The goal of virtualzing 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.

- 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.

- The flow label is not considered immutable in transit, intermediate devices may change it.

- 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:

- Intermediate devices must not insert extension headers [RFC2460bis].

- Extension headers introduce additional packet overhead which may impact performance.

- Extension headers are not covered by transport checksums (as the addresses would be) nor any other checksum.

- 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] Rijsinghani, 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

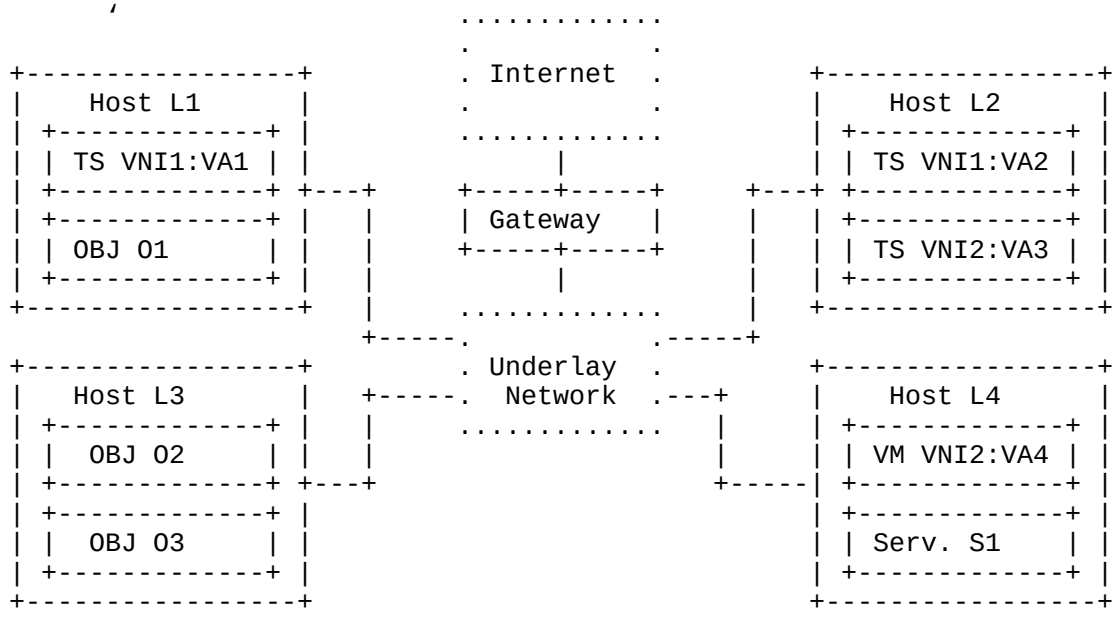
Identifier-locator addressing is broad enough in scope to address many different types of networking entities. For the purposes of this section we classify these as "objects" and "tenant systems".

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*10^-13
10000	4.3368*10^-11
100000	4.3368*10^-09
1000000	4.3368*10^-07

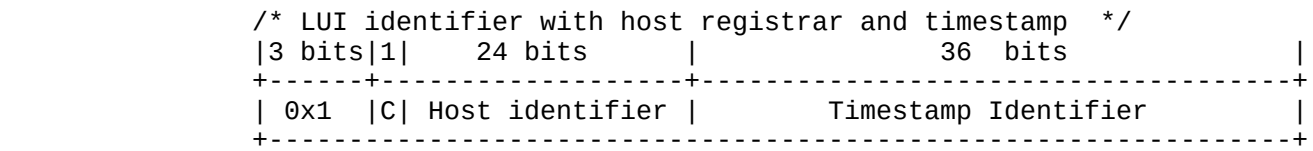
Note that locally unique identifiers may be ephemeral, for instance a task may only exist for a few seconds. This should be considered when determining the probability of identifier collision.

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 be 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.



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 a dramatic simplification since each service can now use a publicly known port number that does not need to 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
Quantonium
4701 Patrick Henry
Santa Clara, CA
EMail: tom@herbertland.com

Petr Lapukhov
1 Hacker Way
Menlo Parck, CA
EMail: petr@fb.com

Internet Area WG
Internet-Draft
Intended status: Standard track
Expires September 14, 2017

T. Herbert
Quantonium
L. Yong
Huawei USA
O. Zia
Microsoft

March 13, 2017

Generic UDP Encapsulation
draft-ietf-intarea-gue-01

Status of this Memo

Copyright Notice

Abstract

This specification describes Generic UDP Encapsulation (GUE), which is a scheme for using UDP to encapsulate packets of 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 as 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

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 the 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 and acronyms

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

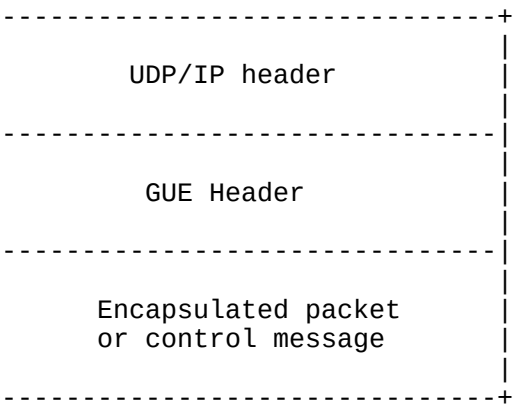
	implicitly addressed to the decapsulator to monitor or control the state or behavior of a tunnel
Flags	A set of bit flags in the primary GUE header
Extension field	An optional field in a GUE header whose presence is indicated by corresponding flag(s)
C-bit	A single bit flag in the primary GUE header that indicates whether the GUE packet contains a control message or data message
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 can be used for private purposes
Outer IP header	Refers to the outer most IP header or packet when encapsulating a packet over IP
Inner IP header	Refers to an encapsulated IP header when an IP packet is encapsulated
Outer packet	Refers to an encapsulating packet
Inner packet	Refers to a packet that is encapsulated

1.2. 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].

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 such as an OAM (Operations, Administration, and Management) 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

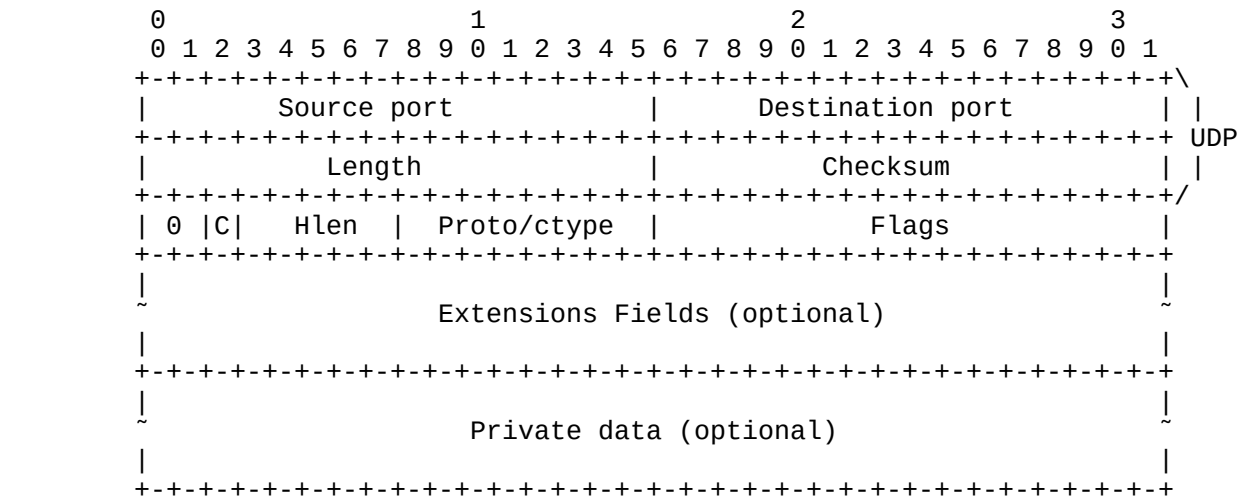
The first two bits of the GUE header contain the GUE protocol version number. The rest of the fields after the GUE version number are defined based on the version number. Versions 0 and 1 are described in this specification; versions 2 and 3 are reserved.

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:

- Source port: If connection semantics (section 5.6.1) are applied to an encapsulation, this is set to the local source port for the connection. When connection semantics are not applied, this is set to a flow entropy value for use with ECMP (Equal-Cost Mulit-Path [RFC2992]). The properties of flow entropy are described in section 5.11.

- 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.

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

- Checksum: Standard UDP checksum (handling is described in section 5.7).

The GUE header consists of:

- Ver: GUE protocol version (0).

- C: C-bit: When set indicates a control message, not set indicates a data message.

- 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 (header_len - 4) / 4 where header_len is the total header length in bytes. All GUE headers are a multiple of four bytes in length. Maximum header length is 128 bytes.

- 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.

- 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.

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

- Private data: Optional private data block (see section 3.4). If the private block is present, it immediately follows that last extension field present in the header. The private block is considered to be part of the GUE 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 fields either contains an Internet protocol number (when the C-bit is not set) or GUE control message type (when the C-bit is set).

3.2.1 Proto field

When the C-bit is not set, the proto/ctype field MUST contain 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). The protocol number serves as an indication of the type of the next protocol header which is contained in the GUE payload at the offset indicated in Hlen. 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.

When the outer IP protocol is IPv4, the proto field MUST be set to a valid IP protocol number usable with IPv4; it MUST NOT be set to a number for 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) also MAY be used with IPv4 as described below.

When the outer IP protocol is IPv6, the proto field can be set to any defined protocol number except that it MUST NOT be set to 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 is processed as though the GUE header is an extension header.

IP protocol number 59 ("No next header") can 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 messages will be defined in an IANA registry. Control message types 1 through 127 may be defined in by RFCs. 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 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. Some flags indicate the presence of an extension fields. The size of an extension field

indicated by a flag MUST be fixed.

Flags can be paired together to allow different lengths for an extension field. For example, if two flag bits are paired, a field can possibly be three different lengths-- that is bit value of 00 indicates no field present; 01, 10, and 11 indicate three possible lengths for the field. Regardless of how flag bits are paired, the lengths and offsets of optional fields corresponding to a set of flags MUST be well defined.

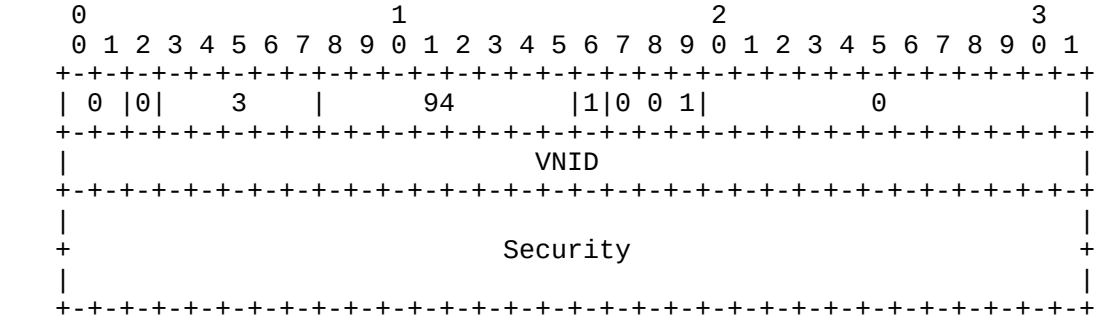
Extension fields are placed in order of the flags. New flags are to be allocated from high to low order bit contiguously without holes. Flags allow random access, for instance to inspect the field corresponding to the Nth flag bit, an implementation only considers the previous N-1 flags to determine the offset. Flags after the Nth flag are not pertinent in calculating the offset of the Nth flag. Random access of flags and fields permits processing of optional extensions in an order that is independent of their position in the packet. The processing order of extensions defined in [GUEEXTENS] demonstrates this property.

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

The set of available flags can be extended in the future by defining a "flag extensions bit" that refers to a field containing an additional set of flags.

3.3.2. Example GUE header with extension fields

An example GUE header for a data message encapsulating an IPv4 packet and containing the VNID and Security extension fields (both defined in [GUEXTENS]) is shown below:



In the above example, the first flag bit is set which indicates that the VNID extension is present this is a 32 bit field. The second through fourth bits of the flags are paired flags that indicate the presence of a security field with eigth possible sizes. In this example 001 indicates a sixty-four bit security field.

3.4. Private data

An implementation MAY use private data for its own use. The private data immediately follows the last 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:

```
Private_length = (Hlen * 4) - Length(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 if present MUST be verified for packet acceptance.

3.5. Message types

3.5.1. Control messages

Control messages carry formatted data 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 can 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.

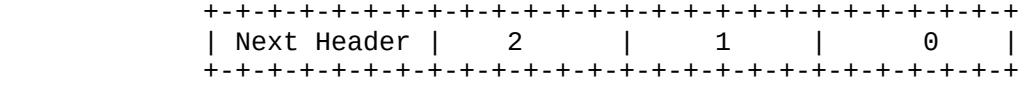
Data messages are indicated in GUE header when the C-bit is not set. The payload of a data message is interpreted as an encapsulated packet of an Internet protocol indicated in the proto/ctype field. The encapsulated packet immediately follows the GUE header.

3.6. Hiding the transport layer protocol number

The GUE header indicates the Internet protocol of an encapsulated packet. A protocol number is either contained in the Proto/ctype field of the primary GUE header or in the Payload Type field of a GUE Transform extension field (used to encrypt the payload with DTLS, [GUEEXTENS). If the transport protocol number needs to be hidden from the network, then a trivial destination options can be used.

The PadN destination option [RFC2460] 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:



For IPv4, it is permitted in GUE to used 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.

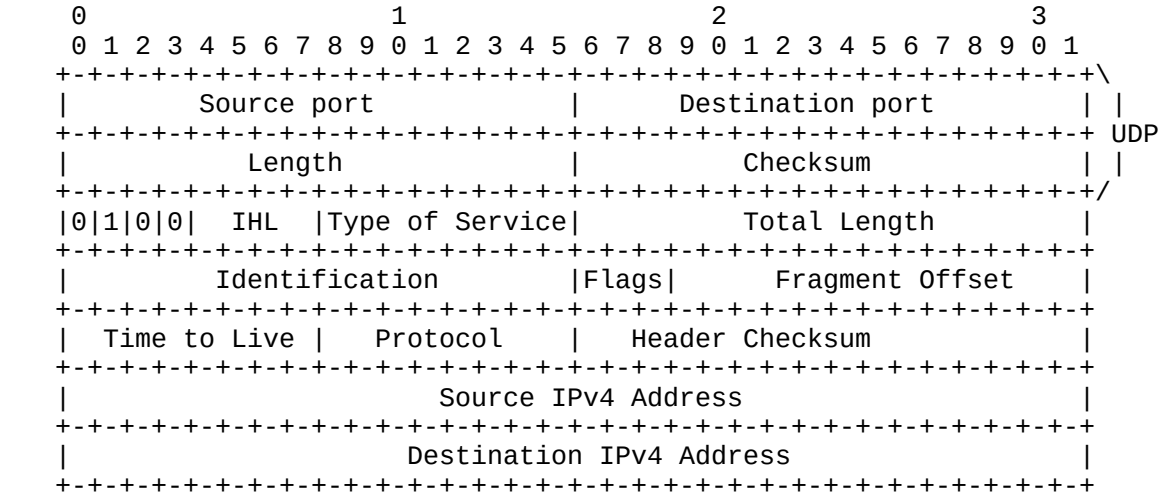
4. Version 1

Version 1 of GUE allows direct encapsulation of IPv4 and IPv6 in UDP. In this version there is no GUE header; a UDP packet carries 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 case 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

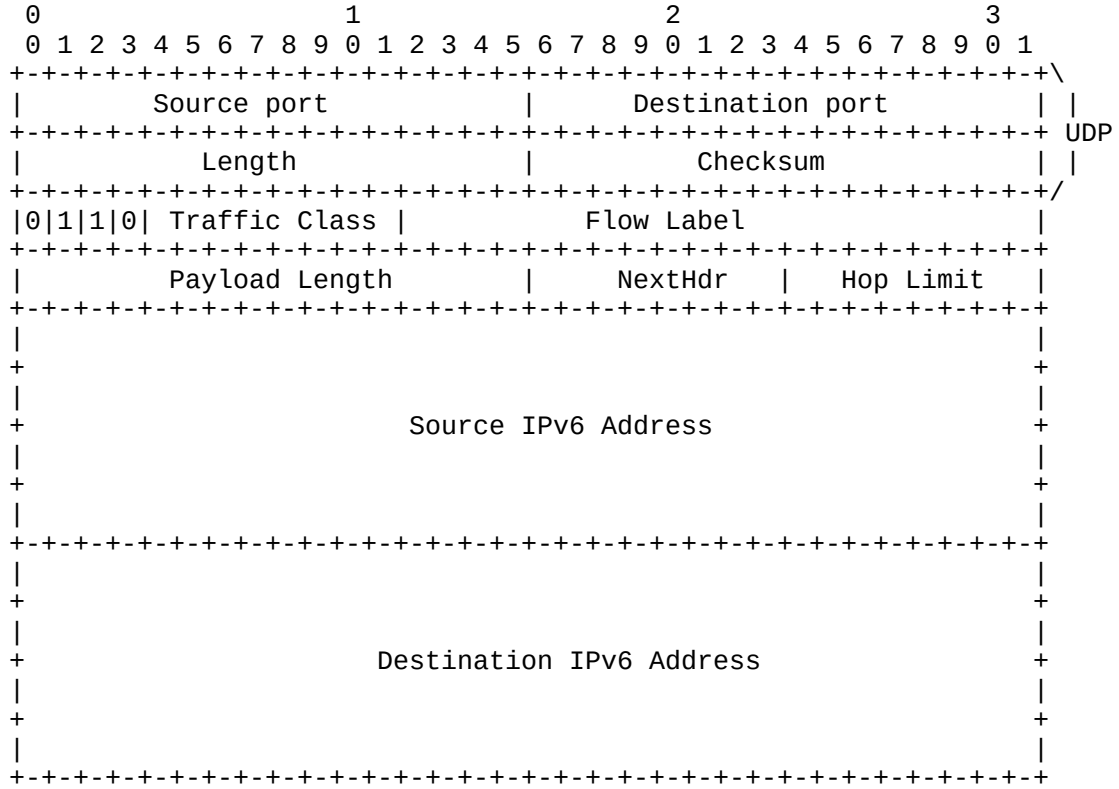
The format for encapsulating IPv4 directly in UDP is:



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

4.2. Direct encapsulation of IPv6

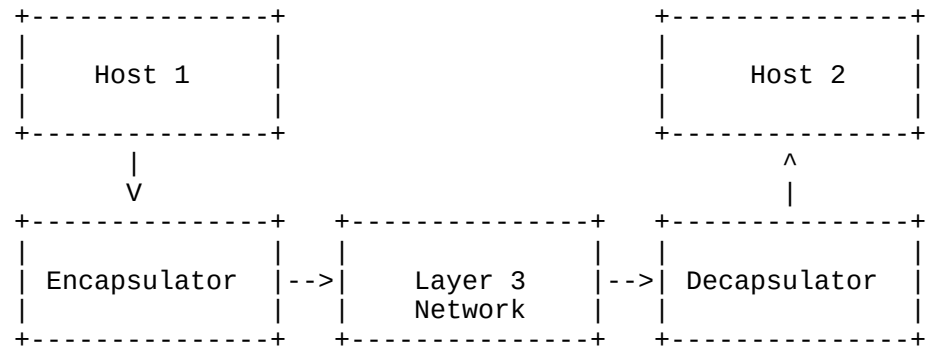
The format for encapsulating IPv6 directly in UDP is demonstrated below:



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. Host 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 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 can be an end host originating the packets of a flow, or can 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, or ESP tunnel mode) -- it SHOULD follow standard conventions for tunneling of one protocol over another. For instance, if an IP packet is being encapsualated in GUE then diffserv interaction [RFC2983] and ECN propagation for tunnels [RFC6040] SHOULD be followed.

5.4. Decapsulator operation

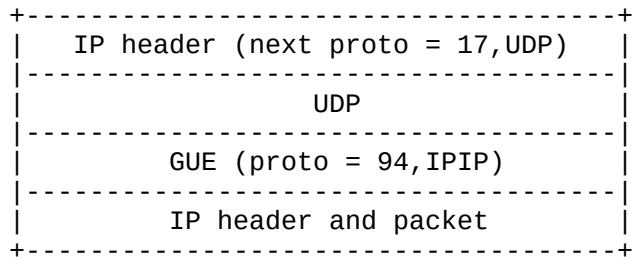
A decapsulator performs decapsulation of GUE packets. A decapsulator is addressed by the outer destination IP address of a GUE packet. The decapsulator validates packets, including fields of the GUE header.

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 payload type, 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 a GUE header 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.

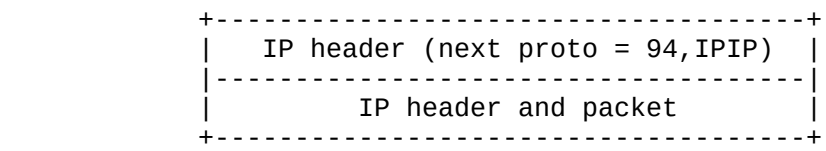
5.4.1. Processing a received data message

If a valid data message is received, the UDP and GUE headers are (logically) removed from the packet. The outer IP header remains intact and the next protocol in the IP 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 does not normally need to parse a GUE header, and none of the flags or extension fields in the GUE header are expected to affect routing.

A router MUST NOT modify a GUE header when forwarding a packet. It MAY encapsulate a GUE packet in another GUE packet, for instance to implement a network tunnel (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 specified provisions to automatically GUE copy flags or 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 merely 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. Inferring connection semantics

A middlebox might infer bidirectional connection semantics for a UDP flow. For instance, a stateful firewall might 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 SHOULD be configured to assume connected semantics defined by the UDP five tuple and the use of GUE encapsulation needs to 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 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 needs to 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, can 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:

- UDP checksums are enabled (for IPv4 or IPv6).
- The GUE header checksum is used (defined in [GUEEXTENS]).
- Use zero UDP checksums. This is always permissible with IPv4; in

IPv6, they can only be used in accordance with applicable requirements in [RFC8086], [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]. Configuration of zero checksums can be selective. For instance, zero checksums might be disallowed 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 or the GUE header checksum SHOULD be used. The UDP checksum and GUE header checksum SHOULD not be used at the same time since that would be mostly redundant.

If neither the UDP checksum or the GUE header checksum is used, then the requirements for using zero IPv6 UDP checksums in [RFC6935] and [RFC6936] MUST be met.

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.

Alternatively 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 that 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 might be warranted.

5.10. Multicast

GUE packets can 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 need to 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) can 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:

- 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.

- 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.

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

[RFC6438] discusses methods to compute and set flow entropy value for IPv6 flow labels. Such 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:

- The value set in the source port is 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.

- The flow entropy has a uniform distribution across encapsulated flows.

- 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).

- 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 encapasulator in creating a flow entropy value. They MAY use the value to match further receive packets for steering decisions, but MUST NOT assume that the hash uniquely or permanently identifies a flow.

- 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.

- 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 need to achieve agreement about GUE parameters will be used in communications. Parameters include GUE version, flags and 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, however the details of implementing these are considered out of scope.

Possible negotiation methods 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. Use of security typically implies 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. Standard optional data such as security, virtual networking identifiers, fragmentation are being defined.
* For extensilbity, GUE uses flag fields as opposed to TLVs as some other encapsulation protocols do. Flag fields are strictly ordered, allow random access, and an efficient use of header

space.

* 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 [RFC7510], and Generic UDP Encapsulation for IP Tunneling (GRE over UDP)[RFC8086]. Generic UDP tunneling [GUT] is a proposal similar to GUE in that it aims to tunnel packets of IP protocols over UDP.

GUE has the following discriminating features:

- UDP encapsulation leverages specialized network device processing for efficient transport. The semantics for using the UDP source port for flow entropy as input to ECMP are defined in section 5.11.

- GUE permits encapsulation of arbitrary IP protocols, which includes layer 2 3, and 4 protocols.

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

with minimal overhead (four bytes of additional header).

- o GUE is extensible. New flags and 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 example, 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 contrast, the number of TCAM entries needed to parse a set of N arbitrarily ordered TLVS is approximately e*N!.

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.

GUE security is provided by extensions for security defined in [GUEEXTENS]. These extensions include methods to authenticate the GUE header and encrypt the GUE payload.

The GUE header can be authenticated using a security extension for an HMAC. Securing the GUE payload can be accomplished use of the GUE Payload Transform that can provide DTLS [RFC6347] in the payload of a GUE packet to encrypt the payload.

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

8. IANA Considerations

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 in accordance with RFC Required policy [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 in accordance with RFC Required policy [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 in accordance with RFC Required policy [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, Bob Briscoe, and Murray Kucherawy 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>.

[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>.

[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>.

[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>.

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <http://www.rfc-editor.org/info/rfc6335>.

10.2. Informative References

[RFC2992] Hopps, C., "Analysis of an Equal-Cost Multi-Path Algorithm", RFC 2992, DOI 10.17487/RFC2992, November 2000, <http://www.rfc-editor.org/info/rfc2992>.

[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>.

[RFC5245] 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>.

[RFC8086] Yong, L., Ed., Crabbe, E., Xu, X., and T. Herbert, "GRE-in-UDP Encapsulation", RFC 8086, DOI 10.17487/RFC8086, March 2017, <http://www.rfc-editor.org/info/rfc8086>.

[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>.

[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>.

[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>.

[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>.

[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>.

[RFC2003] Perkins, C., "IP Encapsulation within IP", RFC 2003, DOI 10.17487/RFC2003, October 1996, <http://www.rfc-editor.org/info/rfc2003>.

[RFC2473] Conta, A. and S. Deering, "Generic Packet Tunneling in IPv6 Specification", RFC 2473, DOI 10.17487/RFC2473, December 1998, <http://www.rfc-editor.org/info/rfc2473>.

[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>.

[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>.

[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

[TOU] Herbert, T., "Transport layer protocols over UDP" draft-herbert-transports-over-udp-00

[CIRCBRK] Fairhurst, G., "Network Transport Circuit Breakers",

draft-ietf-tsvwg-circuit-breaker-15

[TCPUDP] Chesire, S., Graessley, J., and McGuire, R., "Encapsulation of TCP and other Transport Protocols over UDP" draft-cheshire-tcp-over-udp-00

[GUT] Manner, J., Varia, N., and Briscoe, B., "Generic UDP Tunnelling (GUT) draft-manner-tsvwg-gut-02.txt"

[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.

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 selects 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 is 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 is mostly avoided by fragmenting packets before entering a tunnel, GUE fragmentation, path MTU discovery in higher layer protocols, or operator adjusting MTUs. Other UDP traffic might not implement such procedures to avoid fragmentation, so enabling UDP RSS support in the NIC might 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 are of interest: the encapsulated packet's transport checksum, and the UDP checksum in the outer header.

A.2.1. Transmit checksum offload

NICs can 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 (greater than 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 do not contain values that need to 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

B.1. Priveleged 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 could 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 can store a flow hash in its PCB for an inner flow, and might 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

Low level data path protocol, such is GUE, are often supported in high speed network device hardware. 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
Quantonium
4701 Patrick Henry
Santa Clara, CA 95054
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

Internet Area WG
Internet Draft
Intended status: Informational
Updates: 4459
Expires: September 2017

J. Touch
USC/ISI
M. Townsley
Cisco
March 13, 2017

IP Tunnels in the Internet Architecture
draft-ietf-intarea-tunnels-04.txt

Status of this Memo

Abstract

This document discusses the role of IP tunnels in the Internet architecture. An IP tunnel transits IP datagrams as payloads in non-link layer protocols. This document explains the relationship of IP tunnels to existing protocol layers and the challenges in supporting IP tunneling, based on the equivalence of tunnels to links. The implications of this document are used to derive recommendations that update MTU and fragment issues in RFC 4459.

Table of Contents

1. Introduction

The Internet layering architecture is loosely based on the ISO seven layer stack, in which data units traverse the stack by being wrapped inside data units of the next layer down [Cl88][Zi80]. A tunnel is a mechanism for transmitting data units between endpoints by wrapping them as data units of the same or higher layers, e.g., IP in IP (Figure 1) or IP in UDP (Figure 2).

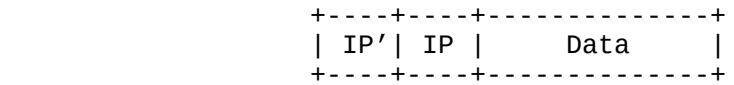


Figure 1 IP inside IP

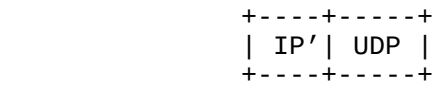


Figure 2 IP in UDP in IP in Ethernet

This document focuses on tunnels that transit IP packets, i.e., in which an IP packet is the payload of another protocol, other than a typical link layer. A tunnel is a virtual link that can help decouple the network topology seen by transiting packets from the underlying physical network [To98][RFC2473]. Tunnels were critical in the development of multicast because not all routers were capable of processing multicast packets [Er94]. Tunnels allowed multicast packets to transit efficiently between multicast-capable routers over paths that did not support native link-layer multicast. Similar techniques have been used to support incremental deployment of other protocols over legacy substrates, such as IPv6 [RFC2546].

Use of tunnels is common in the Internet. The word "tunnel" occurs in nearly 1,500 RFCs (of nearly 8,000 current RFCs, close to 20%), and is supported within numerous protocols, including:

- IP in IP / mobile IP - IPv4 in IPv4 tunnels [RFC2003][RFC2473][RFC5944]
- IP in IPv6 - IPv6 or IPv4 in IPv6 [RFC2473]
- IPsec - includes a tunnel mode to enable encryption or authentication of the an entire IP datagram inside another IP datagram [RFC4301]
- Generic Router Encapsulation (GRE) - a shim layer for tunneling any network layer in any other network layer, as in IP in GRE in IP [RFC2784][RFC7588][RFC7676], or inside UDP in IP [RFC8086]
- MPLS - a shim layer for tunneling IP over a circuit-like path over a link layer [RFC3031] or inside UDP in IP [RFC7510], in which identifiers are rewritten on each hop, often used for traffic provisioning
- LISP - a mechanism that uses multipoint IP tunnels to reduce routing table load within an enclave of routers at the expense of more complex tunnel ingress encapsulation tables [RFC6830]
- TRILL - a mechanism that uses multipoint L2 tunnels to enable use of L3 routing (typically IS-IS) in an enclave of Ethernet bridges [RFC5556][RFC6325]

- Generic UDP Encapsulation (GUE) - IP in UDP in IP [He16]
- Automatic Multicast Tunneling (AMT) - IP in UDP in IP for multicast [RFC7450]
- L2TP - PPP over IP, to extend a subscriber's DSL/FTTH connection from an access line provider to an ISP [RFC3931]
- L2VPNs - provides a link topology different from that provided by physical links [RFC4664]; many of these are not classical tunnels, using only tags (Ethernet VLAN tags) rather than encapsulation
- L3VPNs - provides a network topology different from that provided by ISPs [RFC4176]
- NVO3 - data center network sharing (to be determined, which may include use of GUE or other tunnels) [RFC7364]
- PWE3 - emulates wire-like services over packet-switched services [RFC3985]
- SEAL/AERO -IP in IP tunneling with an additional shim header designed to overcome the limitations of RFC2003 [RFC5320][Te16]

The variety of tunnel mechanisms raises the question of the role of tunnels in the Internet architecture and the potential need for these mechanisms to have similar and predictable behavior. In particular, the ways in which packet sizes (i.e., Maximum Transmission Unit or MTU) mismatch and error signals (e.g., ICMP) are handled may benefit from a coordinated approach.

Regardless of the layer in which encapsulation occurs, tunnels emulate a link. The only difference is that a link operates over a physical communication channel, whereas a tunnel operates over other software protocol layers. Because tunnels are links, they are subject to the same issues as any link, e.g., MTU discovery, signaling, and the potential utility of native support for broadcast and multicast [RFC3819]. Tunnels have some advantages over native links, being potentially easier to reconfigure and control because they can generally rely on existing out-of-band communication between its endpoints.

The first attempt to use large-scale tunnels was to transit multicast traffic across the Internet in 1988, and this resulted in 'tunnel collapse'. At the time, tunnels were not implemented as encapsulation-based virtual links, but rather as loose source routes on un-encapsulated IP datagrams [RFC1075]. Then, as now, routers did

not support use of the loose source route IP option at line rate, and the multicast traffic caused overload of the so-called "slow path" processing of IP datagrams in software. Using encapsulation tunnels avoided that collapse by allowing the forwarding of encapsulated packets to use the "fast path" hardware processing [Er94].

The remainder of this document describes the general principles of IP tunneling and discusses the key considerations in the design of any protocol that tunnels IP datagrams. It derives its conclusions from the equivalence of tunnels and links and from requirements of existing standards for supporting IPv4 and IPv6 as payloads.

2. Conventions used in this document

2.1. Key Words

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].

In this document, these key words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying RFC-2119 significance.

2.2. Terminology

This document uses the following terminology. Optional words in the term are indicated in parentheses, e.g., "(link or network) interface" or "egress (interface)".

Terms from existing RFCs:

- Messages: variable length data labeled with globally-unique endpoint IDs, also known as a datagram for IP messages [RFC791].

- Node: a physical or logical network device that participates as either a host [RFC1122][RFC6434] or router [RFC1812]. This term originally referred to gateways since some very early RFCs [RFC5], but is currently the common way to describe a point in a network at which messages are processed.

- Host or endpoint: a node that sources or sinks messages labeled from/to its IDs, typically known as a host for both IP and higher-layer protocol messages [RFC1122].

- Source or sender: the node that generates a message [RFC1122].

- Destination or receiver: the node that consumes a message [RFC1122].

- Router or gateway: a node that relays IP messages using destination IDs and local context [RFC1812]. Routers also act as hosts when they source or sink messages. Also known as a forwarder for IP messages. Note that the notion of router is relative to the layer at which message processing is considered [To16].

- Link: a communications medium (or emulation thereof) that transfers IP messages between nodes without traversing a router (as would require decrementing the hop count) [RFC1122][RFC1812].

- (Link or network) Interface: a location on a link co-located with a node where messages depart onto that link or arrive from that link. On physical links, this interface formats the message for transmission and interprets the received signals.

- Path: a sequence of one or more links over which an IP message traverses between source and destination nodes (hosts or routers).

- (Link) MTU: the largest message that can transit a link [RFC791], also often referred to simply as "MTU". It does not include the size of link-layer information, e.g., link layer headers or trailers, i.e., it refers to the message that the link can carry as a payload rather than the message as it appears on the link. This is thus the largest network layer packet (including network layer headers, e.g., IP datagram) that can transit a link. Note that this need not be the native size of messages on the link, i.e., the link may internally fragment and reassemble messages. For IPv4, the smallest MTU must be at least 68 bytes [RFC791], and for IPv6 the smallest MTU must be at least 1280 bytes [RFC2460].

- EMTU_S (effective MTU for sending): the largest message that can transit a link, possibly also accounting for fragmentation that happens before the fragments are emitted onto the link [RFC1122]. When source fragmentation is not possible, EMTU_S = (link) MTU. For IPv4, this is MUST be at least 68 bytes [RFC791] and for IPv6 this MUST be at least 1280 bytes [RFC2460].

- EMTU_R (effective MTU to receive): the largest payload message that a receiver must be able to accept. This thus also represents the largest message that can traverse a link, taking into account reassembly at the receiver that happens after the fragments are received [RFC1122]. For IPv4, this is MUST be at least 576 bytes [RFC791] and for IPv6 this MUST be at least 1500 bytes [RFC2460].

- Path MTU (PMTU): the largest message that can transit a path of links [RFC1191][RFC1981]. Typically, this is the minimum of the link MTUs of the links of the path, and represents the largest network layer message (including network layer headers) that can transit a path without requiring fragmentation while in transit. Note that this is not the largest network packet that can be sent between a source and destination, because that network packet might have been fragmented at the network layer of the source and reassembled at the network layer of the destination (if supported).

- Tunnel: a protocol mechanism that transits messages between an ingress interface and egress interface using encapsulation to allow an existing network path to appear as a single link [RFC1853]. Note that a protocol can be used to tunnel itself (IP over IP). There is essentially no difference between a tunnel and the conventional layering of the ISO stack (i.e., by this definition, Ethernet is can be considered tunnel for IP). A tunnel is also known as a virtual link.

- Ingress (interface): the virtual link interface of a tunnel that receives messages within a node, encapsulates them according to the tunnel protocol, and transmits them into the tunnel [RFC2983]. An ingress is the tunnel equivalent of the outgoing (departing) network interface of a link, and its encapsulation processing is the tunnel equivalent of encoding a message for transmission over a physical link. The ingress virtual link interface can be co-located with the traffic source.

  The term ’ingress’ in other RFCs also refers to ’network ingress’, which is the entry point of traffic to a transit network. Because this document focuses on tunnels, the term "ingress" used in the remainder of this document implies "tunnel ingress".

- Egress (interface): a virtual link interface of a tunnel that receives messages that have finished transiting a tunnel and presents them to a node [RFC2983]. For reasons similar to ingress, the term ’egress’ will refer to ’tunnel egress’ throughout the remainder of this document. An egress is the tunnel equivalent of the incoming (arriving) network interface of a link and its decapsulation processing is the tunnel equivalent of interpreting a signal received from a physical link. The egress decapsulates messages for further transit to the destination. The egress virtual link interface can be co-located with the traffic destination.

- Ingress node: network device on which an ingress is attached as a virtual link interface [RFC2983]. Note that a node can act as both an ingress node and an egress node at the same time, but typically only for different tunnels.

- Egress node: device where an egress is attached as a virtual link interface [RFC2983]. Note that a device can act as both a ingress node and an egress node at the same time, but typically only for different tunnels.

- Inner header: the header of the message as it arrives to the ingress [RFC2003].

- Outer header(s): the headers added to the message by the ingress, as part of the encapsulation for tunnel transit [RFC2003].

- Mid-tunnel fragmentation: Fragmentation of the message during the tunnel transit, as could occur for IPv4 datagrams with DF=0 [RFC2983].

- Atomic packet or datagram: an IP packet that has not been fragmented and which cannot be fragmented further [RFC6864]

The following terms are introduced by this document:

- (Tunnel) transit packet: the packet arriving at a node connected to a tunnel that enters the ingress interface and exits the egress interface, i.e., the packet carried over the tunnel. This is sometimes known as the ’tunneled packet’, i.e., the packet carried over the tunnel. This is the tunnel equivalent of a network layer packet as it would traverse a link. This document focuses on IPv4 and IPv6 transit packets.

- (Tunnel) link packet: packets that traverse from ingress interface to egress interface, in which resides all or part of a transit packet. This is the tunnel equivalent of a link layer packet as it would traverse a link, which is why we use the same terminology.

- Tunnel MTU: the largest transit packet that can traverse a tunnel, i.e., the tunnel equivalent of a link MTU, which is why we use the same terminology. This is the largest transit packet which can be reassembled at the egress interface.

- Tunnel atom: the largest transit packet that can traverse a tunnel as an atomic packet, i.e., without requiring tunnel link packet fragmentation either at the ingress or on-path between the ingress and egress.

- Inner fragmentation: fragmentation of the transit packet that arrives at the ingress interface before any additional headers are added. This can only correctly occur for IPv4 DF=0 datagrams.

- Outer fragmentation: source fragmentation of the tunnel link packet after encapsulation; this can involve fragmenting the outermost header or any of the other (if any) protocol layers involved in encapsulation.

- Maximum frame size (MFS): the link-layer equivalent of the MTU, using the OSI term 'frame'. For Ethernet, the MTU (network packet size) is 1500 bytes but the MFS (link frame size) is 1518 bytes originally, and 1522 bytes assuming VLAN (802.1Q) tagging support.

- EMFS_S: the link layer equivalent of EMTU_S.

- EMFS_R: the link layer equivalent of EMTU_R.

- Path MFS: the link layer equivalent of PMTU.

3. The Tunnel Model

A network architecture is an abstract description of a distributed communications system, its components and their relationships, the requisite properties of those components and the emergent properties of the system that result [To03]. Such descriptions can help explain behavior, as when the OSI seven-layer model is used as a teaching example [Zi80]. Architectures describe capabilities - and, just as importantly, constraints.

A network can be defined as a system of endpoints and relays interconnected by communication paths, abstracting away issues of naming in order to focus on message forwarding. To the extent that the Internet has a single, coherent interpretation, its architecture is defined by its core protocols (IP [RFC791], TCP [RFC793], UDP [RFC768]) whose messages are handled by hosts, routers, and links [Cl88][To03], as shown in Figure 3:

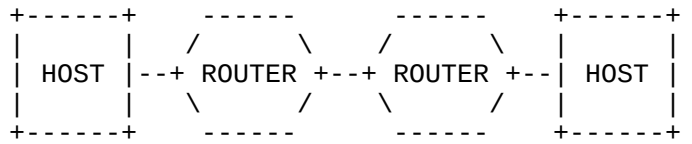


Figure 3 Basic Internet architecture

As a network architecture, the Internet is a system of hosts (endpoints) and routers (relays) interconnected by links that exchange messages when possible. "When possible" defines the Internet's "best effort" principle. The limited role of routers and links represents the End-to-End Principle [Sa84] and longest-prefix match enables hierarchical forwarding using compact tables.

Although the definitions of host, router, and link seem absolute, they are often relative as viewed within the context of one protocol layer, each of which can be considered a distinct network architecture. An Internet gateway is an OSI Layer 3 router when it transits IP datagrams but it acts as an OSI Layer 2 host as it sources or sinks Layer 2 messages on attached links to accomplish this transit capability. In this way, one device (Internet gateway) behaves as different components (router, host) at different layers.

Even though a single device may have multiple roles - even concurrently - at a given layer, each role is typically static and determined by context. An Internet gateway always acts as a Layer 2 host and that behavior does not depend on where the gateway is viewed from within Layer 2. In the context of a single layer, a device's behavior is typically modeled as a single component from all viewpoints in that layer (with some notable exceptions, e.g., Network Address Translators, which appear as hosts and routers, depending on the direction of the viewpoint [To16]).

3.1. What is a Tunnel?

A tunnel can be modeled as a link in another network [To98][To01][To03]. In Figure 4, a source host (Hsrc) and destination host (Hdst) communicating over a network M in which two routers (Ra and Rd) are connected by a tunnel. Keep in mind that it is possible that both network N and network M can both be components of the Internet, i.e., there may be regular traffic as well as tunneled traffic over any of the routers shown.

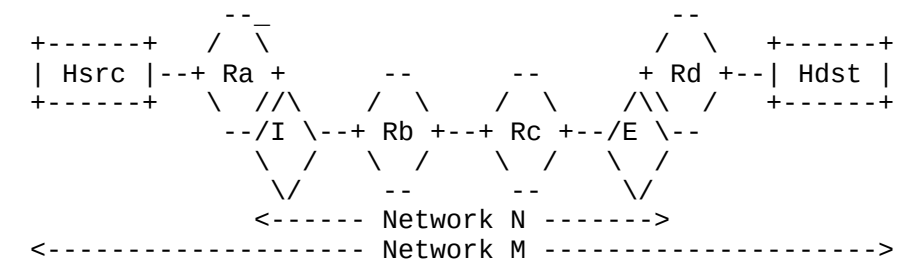


Figure 4 The big picture

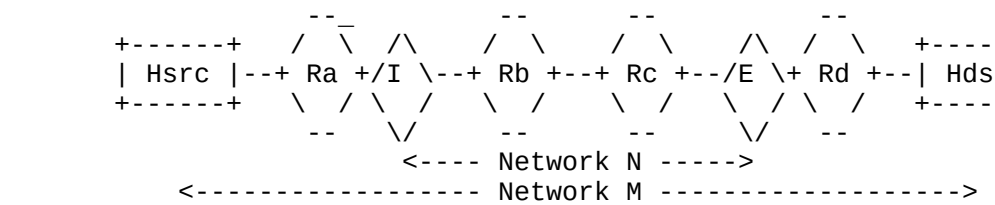
The tunnel consists of two interfaces - an ingress (I) and an egress (E) that lie along a path connected by network N. Regardless of how the ingress and egress interfaces are connected, the tunnel serves as a link between the nodes it connects (here, Ra and Rd).

IP packets arriving at the ingress interface are encapsulated to traverse network N. We call these packets ’tunnel transit packets’ (or just ’transit packets’) because they will transit the tunnel inside one or more of what we call ’tunnel link packets’. Transit packets correspond to network (IP) packets traversing a conventional link and tunnel link packets correspond to the packets of a conventional link layer (which can be called just ’link packets’).

Link packets use the source address of the ingress interface and the destination address of the egress interface - using whatever address is appropriate to the Layer at which the ingress and egress interfaces operate (Layer 2, Layer 3, Layer 4, etc.). The egress interface decapsulates those messages, which then continue on network M as if emerging from a link. To transit packets and to the routers the tunnel connects (Ra and Rd), the tunnel acts as a link and the ingress and egress interfaces act as network interfaces to that link.

The model of each component (ingress and egress interfaces) and the entire system (tunnel) depends on the layer from which they are viewed. From the perspective of the outermost hosts (Hsrc and Hdst), the tunnel appears as a link between two routers (Ra and Rd). For routers along the tunnel (e.g., Rb and Rc), the ingress and egress interfaces appear as the endpoint hosts on network N.

When the tunnel network (N) is implemented using the same protocol as the endpoint network (M), the picture looks flatter (Figure 5), as if it were running over a single network. However, this appearance is incorrect - nothing has changed from the previous case. From the perspective of the endpoints, Rb and Rc and network N don’t exist and aren’t visible, and from the perspective of the tunnel, network M doesn’t exist. The fact that network N and M use the same protocol, and may traverse the same links is irrelevant.



3.2. View from the Outside

As already observed, from outside the tunnel, to network M, the entire tunnel acts as a link (Figure 6). Consequently all requirements for links supporting IP also apply to tunnels [RFC3819].

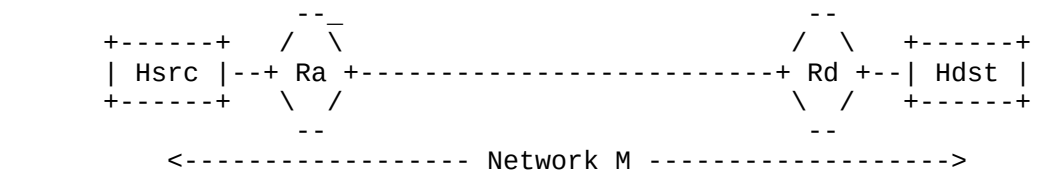


Figure 6 Tunnels as viewed from the outside

For example, the IP datagram hop counts (IPv4 Time-to-Live [RFC791] and IPv6 Hop Limit [RFC2460]) are decremented when traversing a router, but not when traversing a link - or thus a tunnel. Similarly, because the ingress and egress are interfaces on this outer network, they should never issue ICMP messages. A router or host would issue the appropriate ICMP, e.g., "packet too big" (IPv4 fragmentation needed and DF set [RFC792] or IPv6 packet too big [RFC4443]), when trying to send a packet to the egress, as it would for any interface.

Tunnels have a tunnel MTU - the largest message that can transit that tunnel, just as links have a link MTU. Tis MTU may not reflect the native message size of hops within a multihop link (or tunnel) and the same is true for a tunnel. In both cases, the MTU is defined by the link’s (or tunnel’s) effective MTU to receive (EMTU_R).

3.3. View from the Inside

Within network N, i.e., from inside the tunnel itself, the ingress interface is a source of tunnel link packets and the egress interface is a sink - so both are viewed as hosts on network N (Figure 7). Consequently [RFC1122] Internet host requirements apply to ingress and egress interfaces when Network N uses IP (and thus the ingress/egress interfaces use IP encapsulation).

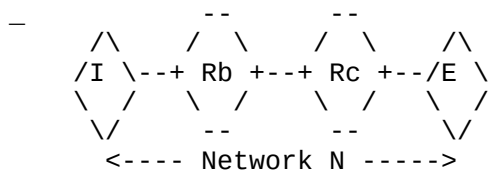


Figure 7 Tunnels, as viewed from within the tunnel

Viewed from within the tunnel, the outer network (M) doesn't exist. Tunnel link packets can be fragmented by the source (ingress interface) and reassembled at the destination (egress interface), just as at conventional hosts. The path between ingress and egress interfaces has a path MTU, but the endpoints can exchange messages as large as can be reassembled at the destination (egress interface), i.e., the EMTU_R of the egress interface. However, in both cases, these MTUs refer to the size of the message that can transit the links and between the hosts of network N, which represents a link layer to network M. I.e., the MTUs of network N represent the maximum frame sizes (MFSs) of the tunnel as a link in network M.

Information about the network - i.e., regarding network N MTU sizes, network reachability, etc. - are relayed from the destination (egress interface) and intermediate routers back to the source (ingress interface), without regard for the external network (M). When such messages arrive at the ingress interface, they may affect the properties of that interface (e.g., its reported MTU to network M), but they should never directly cause new ICMPs in the outer network M. Again, events at interfaces don't generate ICMP messages; it would be the host or router at which that interface is attached that would generate ICMPs, e.g., upon attempting to use that interface.

3.4. Location of the Ingress and Egress

The ingress and egress interfaces are endpoints of the tunnel. Tunnel interfaces may be physical or virtual. The interface may be implemented inside the node where the tunnel attaches, e.g., inside a host or router. The interface may also be implemented as a "bump in the wire" (BITW), somewhere along a link between the two nodes the link interconnects. IP in IP tunnels are often implemented as interfaces on nodes, whereas IPsec tunnels are sometimes implemented as BITW. These implementation variations determine only whether information available at the link endpoints (ingress/egress interfaces) can be easily shared with the connected network nodes.

3.5. Implications of This Model

This approach highlights a few key features of a tunnel as a network architecture construct:

- To the transit packets, tunnels turn a network (Layer 3) path into a (Layer 2) link

- To nodes the tunnel traverses, the tunnel ingress and egress interfaces act as hosts that source and sink tunnel link packets

The consequences of these features are as follow:

- Like a link MTU, a tunnel MTU is defined by the effective MTU of the receiver (i.e., EMTU_R of the egress).

- The messages inside the tunnel are treated like any other link layer, i.e., the MTU is determined by the largest (transit) payload that traverses the link.

- The tunnel path MFS is not relevant to the transited traffic. There is no mechanism or protocol by which it can be determined.

- Because routers, not links, alter hop counts [RFC1812], hopcounts are not decremented solely by the transit of a tunnel. A packet with a hop count of zero should successfully transit a link (and thus a tunnel) that connects two hosts.

- The addresses of a tunnel ingress and egress interface correspond to link layer addresses to the transit packet. Like links, some tunnels may not have their own addresses. Like network interfaces, ingress and egress interfaces typically require network layer addresses.

- Like network interfaces, the ingress and egress interfaces are never a direct source of ICMP messages but may provide information to their attached host or router to generate those ICMP messages during the processing of transit packets.

- Like network interfaces and links, two nodes may be connected by any combination of tunnels and links, including multiple tunnels. As with multiple links, existing network layer forwarding determines which IP traffic uses each link or tunnel.

These observations make it much easier to determine what a tunnel must do to transit IP packets, notably it must satisfy all requirements expected of a link [RFC1122][RFC3819]. The remainder of this document explores these implications in greater detail.

3.6. Fragmentation

There are two places where fragmentation can occur in a tunnel, called ’outer fragmentation’ and ’inner fragmentation’. This document assumes that only outer fragmentation is viable because it is the only approach that works for both IPv4 datagrams with DF=1 and IPv6.

3.6.1. Outer Fragmentation

Outer fragmentation is shown in Figure 8. The bottom of the figure shows the network topology, where transit packets originate at the source, enter the tunnel at the ingress interface for encapsulation, exit the tunnel at the egress interface where they are decapsulated, and arrive at the destination. The packet traffic is shown above the topology, where the transit packets are shown at the top. In this diagram, the ingress interface is located on router 'Ra' and the egress interface is located on router 'Rd'.

When the link packet - which is the encapsulated transit packet - would exceed the tunnel MTU, the packet needs to be fragmented. In this case the packet is fragmented at the outer (link) header, with the fragments shown as (b1) and (b2). The outer header indicates fragmentation (as ' and "), the inner (transit) header occurs only in the first fragment, and the inner (transit) data is broken across the two packets. These fragments are reassembled at the egress interface during decapsulation in step (c), where the resulting link packet is reassembled and decapsulated so that the transit packet can continue on its way to the destination.

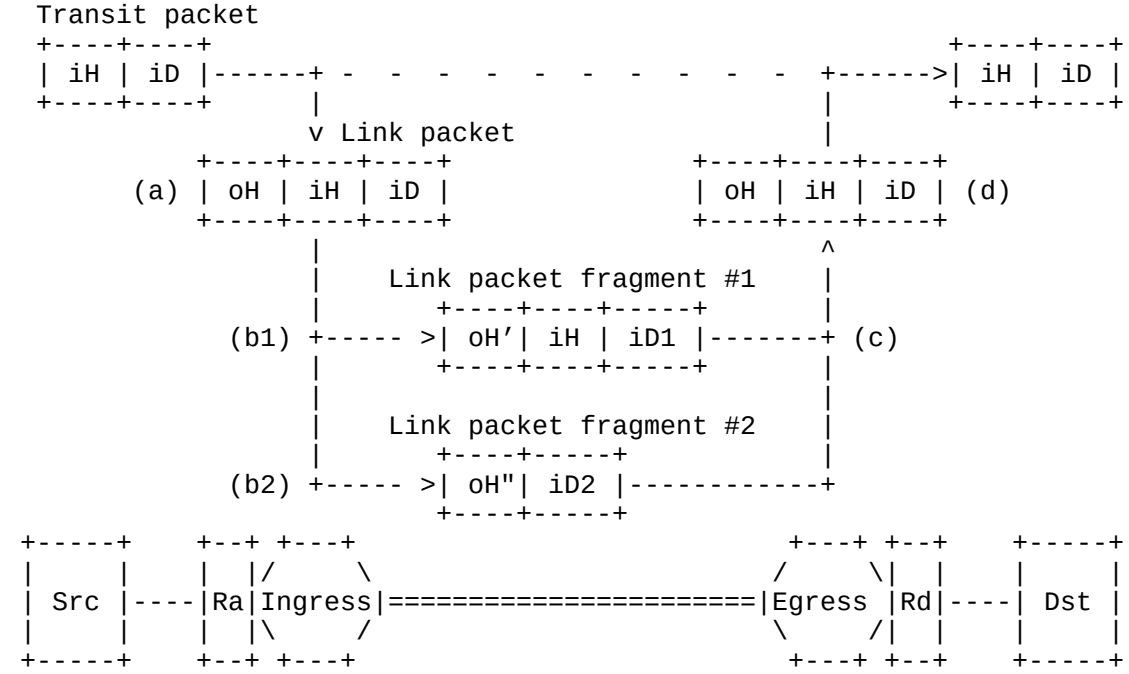


Figure 8 Fragmentation of the (outer) link packet

Outer fragmentation isolates the tunnel encapsulation duties to the ingress and egress interfaces. This can be considered a benefit in clean, layered network design, but also may require complex egress interface decapsulation, especially where tunnels aggregate large amounts of traffic, such as may result in IP ID overload (see Sec. 4.1.4). Outer fragmentation is valid for any tunnel link protocol that supports fragmentation (e.g., IPv4 or IPv6), in which the tunnel endpoints act as the host endpoints of that protocol.

Along the tunnel, the inner (transit) header is contained only in the first fragment, which can interfere with mechanisms that 'peek' into lower layer headers, e.g., as for relayed ICMP (see Sec. 4.3).

3.6.2. Inner Fragmentation

Inner fragmentation distributes the impact of tunnel fragmentation across both egress interface decapsulation and transit packet destination, as shown in Figure 9; this can be especially important when the tunnel would otherwise need to source (outer) fragment large amounts of traffic. However, this mechanism is valid only when the transit packets can be fragmented on-path, e.g., as when the transit packets are IPv4 datagrams with DF=0.

Again, the network topology is shown at the bottom of the figure, and the original packets show at the top. Packets arrive at the ingress node (router Ra) and are fragmented there based into transit packet fragments #1 (a1) and #2 (a2). These fragments are encapsulated at the ingress interface in steps (b1) and (b2) and each resulting link packet traverses the tunnel. When these link packets arrive at the egress interface they are decapsulated in steps (c1) and (c2) and the egress node (router) forwards the transit packet fragments to their destination. This destination is then responsible for reassembling the transit packet fragments into the original transit packet (d).

Along the tunnel, the inner headers are copied into each fragment, and so can be 'peeked at' inside the tunnel (see Sec. 4.3). Fragmentation shifts from the ingress interface to the ingress router and reassembly shifts from the egress interface to the destination.

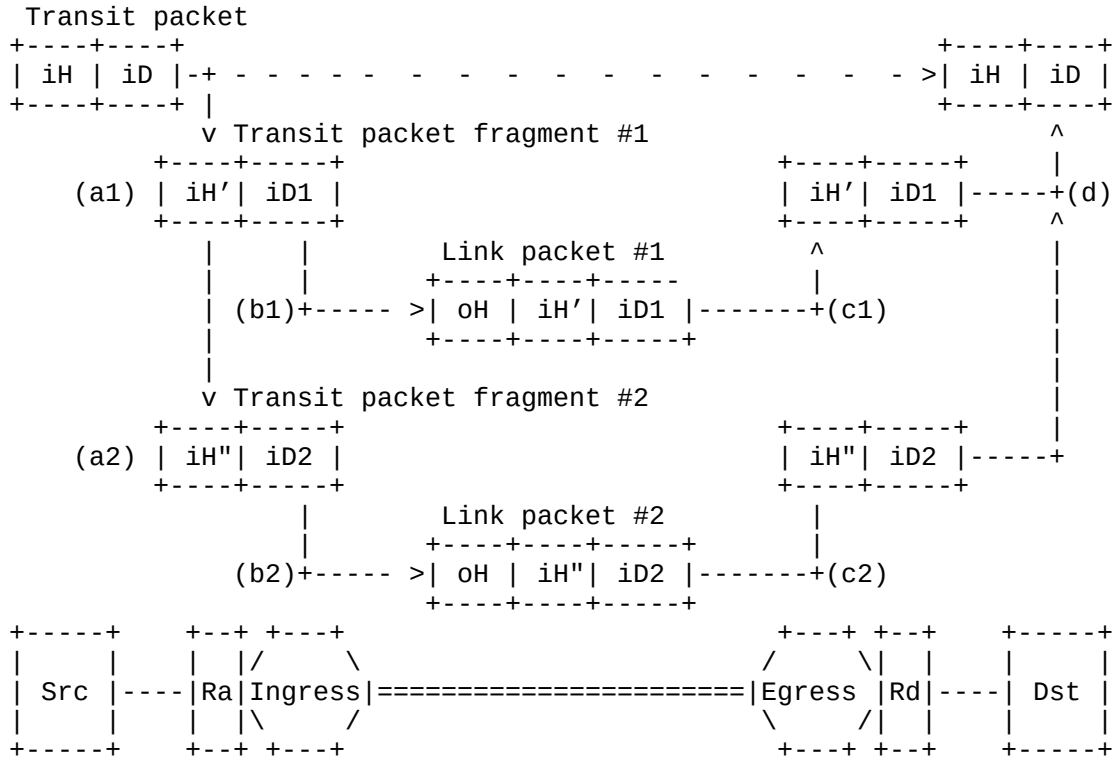


Figure 9 Fragmentation of the inner (transit) packet

3.6.3. The Necessity of Outer Fragmentation

Fragmentation is critical for tunnels that support transit packets for protocols with minimum MTU requirements, while operating over tunnel paths using protocols that have their own MTU requirements. Depending on the amount of space used by encapsulation, these two minimums will ultimately interfere (especially when a protocol transits itself either directly, as with IP-in-IP, or indirectly, as in IP-in-GRE-in-IP), and the transit packet will need to be fragmented to both support a tunnel MTU while traversing tunnels with their own tunnel path MTUs.

Outer fragmentation is the only solution that supports all IPv4 and IPv6 traffic, because inner fragmentation is allowed only for IPv4 datagrams with DF=0.

4. IP Tunnel Requirements

The requirements of an IP tunnel are defined by the requirements of an IP link because both transit IP packets. A tunnel thus must transit the IP minimum MTU, i.e., 68 bytes for IPv4 [RFC793] and 1280 bytes for IPv6 [RFC2460] and a tunnel must support address resolution when there is more than one egress interface for that tunnel.

The requirements of the tunnel ingress and egress interfaces are defined by the network over which they exchange messages (link packets). For IP-over-IP, this means that the ingress interface MUST NOT exceed the IP fragment identification field uniqueness requirements [RFC6864]. Uniqueness is more difficult to maintain at high packet rates for IPv4, whose fragment ID field is only 16 bits.

These requirements remain even though tunnels have some unique issues, including the need for additional space for encapsulation headers and the potential for tunnel MTU variation.

4.1. Encapsulation Header Issues

Tunneling uses encapsulation uses a non-link protocol as a link layer. The encapsulation layer thus has the same requirements and expectations as any other IP link layer when used to transit IP packets. These relationships are addressed in the following subsections.

4.1.1. General Principles of Header Fields Relationships

Some tunnel specifications attempt to relate the header fields of the transit packet and tunnel link packet. In some cases, this relationship is warranted, whereas in other cases the two protocol layers need to be isolated from each other. For example, the tunnel link header source and destination addresses are network endpoints in the tunnel network N, but have no meaning in the outer network M. The two sets of addresses are effectively independent, just as are other network and link addresses.

Because the tunneled packet uses source and destination addresses with a separate meaning, it is inappropriate to copy or reuse the IPv4 Identification (ID) or IPv6 Fragment ID fields of the tunnel transit packet (see Section 4.1.4). Similarly, the DF field of the transit packet is not related to that field in the tunnel link packet header (presuming both are IPv4) (see Section 4.2). Most other fields are similarly independent between the transit packet and tunnel link packet. When a field value is generated in the encapsulation header, its meaning should be derived from what is desired in the context of

the tunnel as a link. When feedback is received from these fields, they should be presented to the tunnel ingress and egress as if they were network interfaces. The behavior of the node where these interfaces attach should be identical to that of a conventional link.

There are exceptions to this rule that are explicitly intended to relay signals from inside the tunnel to the network outside the tunnel, typically relevant only when the tunnel network N and the outer network M use the same network. These apply only when that coordination is defined, as with explicit congestion notification (ECN) [RFC6040] (see Section 4.3.2), and differentiated services code points (DSCPs) [RFC2983]. Equal-cost multipath routing may also affect how some encapsulation fields are set, including IPv6 flow labels [RFC6438] and source ports for transport protocols when used for tunnel encapsulation [RFC8085] (see Section 4.3.4).

4.1.2. Addressing Fields

Tunnel ingresses and egresses have addresses associated with the encapsulation protocol. These addresses are the source and destination (respectively) of the encapsulated packet while traversing the tunnel network.

Tunnels may or may not have addresses in the network whose traffic they transit (e.g., network M in Figure 4). In some cases, the tunnel is an unnumbered interface to a point-to-point virtual link. When the tunnel has multiple egresses, tunnel interfaces require separate addresses in network M.

To see the effect of tunnel interface addresses, consider traffic sourced at router Ra in Figure 4. Even before being encapsulated by the ingress, traffic needs a source IP network address that belongs to the router. One option is to use an address associated with one of the other interfaces of the router [RFC1122]. Another option is to assign a number to the tunnel interface itself. Regardless of which address is used, the resulting IP packet is then encapsulated by the tunnel ingress using the ingress address as a separate operation.

4.1.3. Hop Count Fields

The Internet hop count field is used to detect and avoid forwarding loops that cannot be corrected without a synchronized reboot. The IPv4 Time-to-Live (TTL) and IPv6 Hop Limit field each serve this purpose [RFC791][RFC2460]. The IPv4 TTL field was originally intended to indicate packet expiration time, measured in seconds. A router is required to decrement the TTL by at least one or the number of seconds the packet is delayed, whichever is larger [RFC1812]. Packets

are rarely held that long, and so the field has come to represent the count of the number of routers traversed. IPv6 makes this meaning more explicit.

These hop count fields represent the number of network forwarding elements (routers) traversed by an IP datagram. An IP datagram with a hop count of zero can traverse a link between two hosts because it never visits a router (where it would need to be decremented and would have been dropped).

An IP datagram traversing a tunnel thus need not have its hop count modified, i.e., the tunnel transit header need not be affected. A zero hop count datagram should be able to traverse a tunnel as easily as it traverses a link. A router MAY be configured to decrement packets traversing a particular link (and thus a tunnel), which may be useful in emulating a tunnel path as if it were a network path that traversed one or more routers, but this is strictly optional. The ability of the outer network M and tunnel network N to avoid indefinitely looping packets does not rely on the hop counts of the transit packet and tunnel link packet being related.

The hop count field is also used by several protocols to determine whether endpoints are 'local', i.e., connected to the same subnet (link-local discovery and related protocols [RFC4861]). A tunnel is a way to make a remote network address appear directly-connected, so it makes sense that the other ends of the tunnel appear local and that such link-local protocols operate over tunnels unless configured explicitly otherwise. When the interfaces of a tunnel are numbered, these can be interpreted the same way as if they were on the same link subnet.

4.1.4. IP Fragment Identification Fields

Both IPv4 and IPv6 include an IP Identification (ID) field to support IP datagram fragmentation and reassembly [RFC791][RFC1122][RFC2460]. When used, the ID field is intended to be unique for every packet for a given source address, destination address, and protocol, such that it does not repeat within the Maximum Segment Lifetime (MSL).

For IPv4, this field is in the default header and is meaningful only when either source fragmented or DF=0 ("non-atomic packets") [RFC6864]. For IPv6, this field is contained in the optional Fragment Header [RFC2460]. Although IPv6 supports only source fragmentation, the field may occur in atomic fragments [RFC6946].

Although the ID field was originally intended for fragmentation and reassembly, it can also be used to detect and discard duplicate

packets, e.g., at congested routers (see Sec. 3.2.1.5 of [RFC1122]). For this reason, and because IPv4 packets can be fragmented anywhere along a path, all non-atomic IPv4 packets and all IPv6 packets between a source and destination of a given protocol must have unique ID values over the potential fragment reordering period [RFC2460][RFC6864].

The uniqueness of the IP ID is a known problem for high speed nodes, because it limits the speed of a single protocol between two endpoints [RFC4963]. Although this RFC suggests that the uniqueness of the IP ID is moot, tunnels exacerbate this condition. A tunnel often aggregates traffic from a number of different source and destination addresses, of different protocols, and encapsulates them in a header with the same ingress and egress addresses, all using a single encapsulation protocol. If the ingress enforces IP ID uniqueness, this can either severely limit tunnel throughput or can require substantial resources; the alternative is to ignore IP ID uniqueness and risk reassembly errors. Although fragmentation is somewhat rare in the current Internet at large, but it can be common along a tunnel. Reassembly errors are not always detected by other protocol layers (see Sec. 4.3.3) , and even when detected they can result in excessive overall packet loss and can waste bandwidth between the egress and ultimate packet destination.

The 32-bit IPv6 ID field in the Fragment Header is typically used only during source fragmentation. The size of the ID field is typically sufficient that a single counter can be used at the tunnel ingress, regardless of the endpoint addresses or next-header protocol, allowing efficient support for very high throughput tunnels.

The smaller 16-bit IPv4 ID is more difficult to correctly support. A recent update to IPv4 allows the ID to be repeated for atomic packets. When either source fragmentation or on-path fragmentation is supported, the tunnel ingress may need to keep independent ID counters for each tunnel source/destination/protocol tuple.

4.1.5. Checksums

IP traffic transiting a tunnel needs to expect a similar level of error detection and correction as it would expect from any other link. In the case of IPv4, there are no such expectations, which is partly why it includes a header checksum [RFC791].

IPv6 omitted the header checksum because it already expects most link errors to be detected and dropped by the link layer and because it also assumes transport protection [RFC2460]. When transiting IPv6

over IPv6, the tunnel fails to provide the expected error detection. This is why IPv6 is often tunneled over layers that include separate protection, such as GRE [RFC2784].

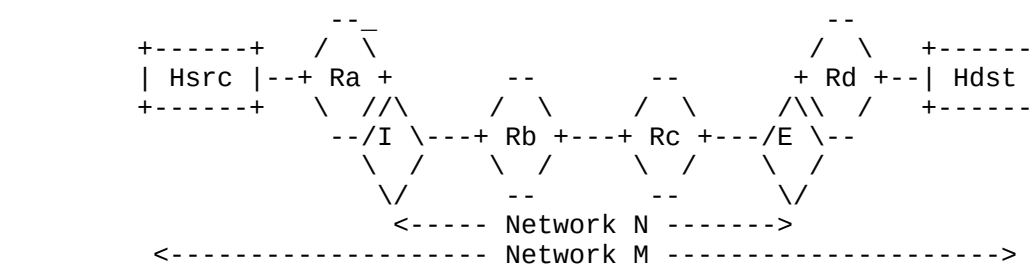
The fragmentation created by the tunnel ingress can increase the need for stronger error detection and correction, especially at the tunnel egress to avoid reassembly errors. The Internet checksum is known to be susceptible to reassembly errors that could be common [RFC4963], and should not be relied upon for this purpose. This is why some tunnel protocols, e.g., SEAL and AERO [RFC5320][Te16], include a separate checksum. This requirement can be undermined when using UDP as a tunnel with no UDP checksum (as per [RFC6935][RFC6936]) when fragmentation occurs because the egress has no checksum with which to validate reassembly. For this reason, it is safe to use UDP with a zero checksum for atomic tunnel link packets only; when used on fragments, whether generated at the ingress or en-route inside the tunnel, omission of such a checksum can result in reassembly errors that can cause additional work (capacity, forwarding processing, receiver processing) downstream of the egress.

4.2. MTU Issues

Link MTUs, IP datagram limits, and transport protocol segment sizes are already related by several requirements [RFC768][RFC791][RFC1122][RFC1812][RFC2460] and by a variety of protocol mechanisms that attempt to establish relationships between them, including path MTU discovery (PMTUD) [RFC1191][RFC1981], packetization layer path MTU discovery (PLMTUD) [RFC4821], as well as mechanisms inside transport protocols [RFC793][RFC4340][RFC4960]. The following subsections summarize the interactions between tunnels and MTU issues, including minimum tunnel MTUs, tunnel fragmentation and reassembly, and MTU discovery.

4.2.1. Minimum MTU Considerations

There are a variety of values of minimum MTU values to consider, both in a conventional network and in a tunnel as a link in that network. These are indicated in Figure 10, an annotated variant of Figure 4. Note that a (link) MTU (a) corresponds to a tunnel MTU (d) and that a path MTU (b) corresponds to a tunnel path MTU (e). The tunnel MTU is the EMTU_R of the egress interface, because that defines the largest transit packet message that can traverse the tunnel as a link in network M. The ability to traverse the hops of the tunnel - in network N - is not related, and only the ingress need be concerned with that value.



Communication in network M viewed at that layer:
(a) <-> Link MTU
(b) <---- Tunnel MTU --------->
(c) <----------- Path MTU ----------------->
(d) <------------------- EMTU_R --------------------------->

Communication in network N viewed at that layer:
(e) <--> Link MTU
(f) <--- Path MTU ------>
(g) <----- EMTU_R --------->

Communication in network N viewed from network M:
(h) <--> MFS
(i) <--- Path MFS ------>
(j) <----- EMFS_R --------->

Figure 10 The variety of MTU values

Consider the following example values. For IPv6 transit packets, the minimum (link) MTU (a) is 1280 bytes, which similarly applies to tunnels as the tunnel MTU (b). The path MTU (c) is the minimum of the links (including tunnels as links) along a path, and indicates the smallest IP message (packet or fragment) that can traverse a path between a source and destination without on-path fragmentation (e.g., supported in IPv4 with DF=0). Path MTU discovery, either at the network layer (PMTUD [RFC1191][RFC1981]) or packetization layer (PLPMTUD [RFC4821]) attempts to tune the source IP packets and fragments (i.e., EMTU_S) to fit within this path MTU size to avoid fragmentation and reassembly [Ke95]. The minimum EMTU_R (c) is 1500 bytes, i.e., the minimum MTU for endpoint-to-endpoint communication.

The tunnel is a source-destination communication in network N. Messages between the tunnel source (the ingress interface) and tunnel destination (egress interface) similarly experience a variety of network N MTU values, including a link MTU (e), a path MTU (f), and an EMTU_R (g). The network N EMTU_S is limited by the path MTU, and the source-destination message maximum is limited by EMTU_R, just as

it was in for those types of MTUs in network M. For an IPv6 network N, its link and path MTUs must be at least 1280 and its EMTU_R must be at least 1500.

However, viewed from the context of network M, these network N MTUs are link layer properties, i.e., maximum frame sizes (MFS). The network N EMTU_R determines the largest message that can transit between the source (ingress) and destination (egress), but viewed from network M this is a link layer, i.e., EMFS_R. The tunnel EMTU_R is EMFS_R minus the link (encapsulation) headers includes the encapsulation headers of the link layer. Just as the path MTU has no bearing on EMTU_R, the path MFS in network N has no bearing on the MTU of the tunnel.

For IPv6 networks M and N, these relationships are summarized as follows:

- Network M MTU = 1280, the largest transit packet (i.e., payload) over a single IPv6 link in the base network without source fragmentation
- Network M path MTU = 1280, the transit packet (i.e., payload) that can traverse a path of links in the base network without source fragmentation
- Network M EMTU_R = 1500, the largest transit packet (i.e., payload) that can traverse a path in the base network with source fragmentation
- Network N MTU = 1280 (for the same reasons as for network M)
- Network N path MTU = 1280 (for the same reasons as for network M)
- Network N EMTU_R = 1500 (for the same reasons as for network M)
- Tunnel MTU = 1500-encapsulation (typically 1460), the network N EMTU_R payload
- Tunnel atom = largest network M message that transits a tunnel using network N as a link layer without fragmentation: 1280-encapsulation, i.e., the network N EMTU_S payload, treating EMTU_S as a network M EMFS_S.

The difference between the network N MTU and its treatment as a link layer in network M is the reason why the tunnel ingress interfaces need to support fragmentation and tunnel egress interfaces need to support reassembly in the encapsulation layer(s). The high cost of fragmentation and reassembly is why it is useful for applications to avoid sending messages too close to the size of the tunnel path MTU [Ke95], although there is no signaling mechanism that can achieve this (see Section 4.2.3).

4.2.2. Fragmentation

A tunnel interacts with fragmentation in two different ways. As a link in network M, transit packets might be fragmented before they reach the tunnel - i.e., in network M either during source fragmentation (if generated at the same node as the ingress interface) or forwarding fragmentation (for IPv4 DF=0 datagrams). In addition, link packets traversing inside the tunnel may require fragmentation by the ingress interface - i.e., source fragmentation by the ingress as a host in network N. These two fragmentation operations are no more related than are conventional IP fragmentation and ATM segmentation and reassembly; one occurs at the (transit) network layer, the other at the (virtual) link layer.

Although many of these issues with tunnel fragmentation and MTU handling were discussed in [RFC4459], that document described a variety of alternatives as if they were independent. This document explains the combined approach that is necessary.

Like any other link, an IPv4 tunnel must transit 68 byte packets without requiring source fragmentation [RFC791][RFC1122] and an IPv6 tunnel must transit 1280 byte packets without requiring source fragmentation [RFC2460]. The tunnel MTU interacts with routers or hosts it connects the same way as would any other link MTU. The pseudocode examples in this section use the following values:

- TP: transit packet
- TPsize: size of the transit packet (including its headers)
- encaps: ingress encapsulation overhead (tunnel link headers)
- tunMTU: tunnel MTU, i.e., network N egress EMTU_R - encaps.
- tunAtom: tunnel atom size, equal to the egress host-level EMTU_S - encaps.

These rules apply at the host/router where the tunnel is attached, i.e., at the network layer of the transit packet (we assume that all tunnels, including multipoint tunnels, have a single, uniform MTU). These are basic source fragmentation rules (or transit refragmentation for IPv4 DF=0 datagrams), and have no relation to the tunnel itself other than to consider the tunnel MTU as the effective link MTU of the next hop.

Inside the source during transit packet generation or a router during transit packet forwarding, the tunnel is treated as if it were any other link (i.e., this is not tunnel processing, but rather typical source or router processing), as indicated in the pseudocode in Figure 11.

```
if (TPsize > tunMTU) then
   if (TP can be on-path fragmented, e.g., IPv4 DF=0) then
      split TP into fragments of tunMTU size
      and send each fragment to the tunnel ingress interface
   else
      drop the TP and send ICMP "too big" to TP source
   endif
else
   send TP to the tunnel ingress
endif
```

Figure 11 Router / host packet size processing algorithm

The tunnel ingress acts as host on the tunnel path, i.e., as source fragmentation of tunnel link packets (we assume that all tunnels, even multipoint tunnels, have a single, uniform tunnel MTU), using the pseudocode shown in Figure 12. Note that ingress source fragmentation occurs in the encapsulation process, which may involve more than one protocol layer. In those cases, fragmentation can occur at any of the layers of encapsulation in which it is supported, based on the configuration of the ingress.

```
if (TPsize <= tunAtom) then
   encapsulate the TP and emit
else
   if (tunAtom < TPsize) then
      fragment TP into tunAtom chunks
      encapslate each chunk and emit
   endif
endif
```

Figure 12 Ingress processing algorithm

Just as a network interface should never receive a message larger than its MTU, a tunnel should never receive a message larger than its tunnel MTU limit (see the host/router processing above). A router attempting to process such a message would already have generated an ICMP "packet too big" and the transit packet would have been dropped before entering into this algorithm. Similarly, a host would have generated an error internally and aborted the attempted transmission.

As an example, consider IPv4 over IPv6 or IPv6 over IPv6 tunneling, where IPv6 encapsulation adds a 40 byte fixed header plus IPv6 options (i.e., IPv6 header extensions) of total size 'EHsize'. The tunnel MTU will be at least 1500 - (40 + EHsize) bytes. The tunnel path MTU will be at least 1280 - (40 + EHsize) bytes. Transit packets larger than 1460-EHsize will be dropped by a node before ingress processing. Considering these minimum values, the previous algorithm uses actual values shown in the pseudocode in Figure 13.

```
if (TPsize <= (1240 - EHsize)) then
   encapsulate TP and emit
else
   if ((1240 - EHsize) < TPsize) then
      fragment TP  into (1240 - EHsize) chunks
      encapsulate each chunk and emit
   endif
endif
```

Figure 13 Ingress processing for an tunnel over IPv6

An IPv6 tunnel supports IPv6 transit only if EHsize is 180 bytes or less; otherwise the incoming transit packet would have been dropped as being too large by the host/router. Similarly, an IPv6 tunnel supports IPv4 transit only if EHsize is 884 bytes or less. In this example, transit packets of up to (1240 - Ehsize) can traverse the tunnel without ingress source fragmentation and egress reassembly.

When using IP directly over IP, the minimum transit packet EMTU_R for IPv4 is 576 bytes and for IPv6 is 1500 bytes. This means that tunnels of IPv4-over-IPv4, IPv4-over-IPv6, and IPv6-over-IPv6 are possible without additional requirements, but this may involve ingress fragmentation and egress reassembly. IPv6 cannot be tunneled directly over IPv4 without additional requirements, notably that the egress EMTU_R is at least 1280 bytes.

When ongoing ingress fragmentation and egress reassembly would be prohibitive or costly, larger MTUs can be supported by design and confirmed either out-of-band (by design) or in-band (e.g., using PLPMTUD [RFC4821], as done in SEAL [RFC5320] and AERO [Te16]).

4.2.3. Path MTU Discovery

Path MTU discovery (PMTUD) enables a network path to support a larger PMTU than it can assume from the minimum requirements of protocol over which it operates. Note, however, that PMTUD never discovers EMTU_R that is larger than the required minimum; that information is available to some upper layer protocols, such as TCP [RFC1122], but cannot be determined at the IP layer.

There is temptation to optimize tunnel traversal so that packets are not fragmented between ingress and egress, i.e., to attempt tune the network M PMTU to the tunnel atom size (i.e., the ingress EMTU_S minus encapsulation overhead) rather than the tunnel MTU, to avoid ingress fragmentation.

This is often impossible because the ICMP "packet too big" message (IPv4 fragmentation needed [RFC792] or IPv6 packet too big [RFC4443]) indicates the complete failure of a link to transit a packet, not a preference for a size that matches that internal the mechanism of the link. ICMP messages are intended to indicate whether a tunnel MTU is insufficient; there is no ICMP message that can indicate when a transit packet is "too bit to for the tunnel path MTU, but not larger than the tunnel MTU". If there were, endpoints might receive that message for IP packets larger than 40 bytes (the payload of a single ATM cell, allowing for the 8-byte AAL5 trailer), but smaller than 9K (the ATM EMTU_R payload).

In addition, attempting to try to tune the network transit size to natively match that of the link internal transit can be hazardous for many reasons:

- The tunnel is capable of transiting packets as large as the network N EMTU_R - encapsulation, which is always at least as large as the tunnel MTU and typically is larger.

- ICMP has only one type of error message regarding large packets - "too big", i.e., too large to transit. There is no optimization message of "bigger than I’d like, but I can deal with if needed".

- IP tunnels often involve some level of recursion, i.e., encapsulation over itself [RFC4459].

Tunnels that use IPv4 as the encapsulation layer SHOULD set DF=0, but this requires generating unique fragmentation ID values, which may limit throughput [RFC6864]. These tunnels might have difficulty assuming ingress EMTU_S values over 64 bytes, so it may not be feasible to assume that larger packets with DF=1 are safe.

Recursive tunneling occurs whenever a protocol ends up encapsulated in itself. This happens directly, as when IPv4 is encapsulated in IPv4, or indirectly, as when IP is encapsulated in UDP which then is a payload inside IP. It can involve many layers of encapsulation because a tunnel provider isn't always aware of whether the packets it transits are already tunneled.

Recursion is impossible when the tunnel transit packets are limited to that of the native size of the ingress payload. Arriving tunnel transit packets have a minimum supported size (1280 for IPv6) and the tunnel PMFS has the same requirement; there would be no room for the tunnel's "link layer" headers, i.e., the encapsulation layer. The result would be an IPv6 tunnel that cannot satisfy IPv6 transit requirements.

It is more appropriate to require the tunnel to satisfy IP transit requirements and enforce that requirement at design time or during operation (the latter using PLPMTUD [RFC4821]). Conventional path MTU discovery (PMTUD) relies on existing endpoint ICMP processing of explicit negative feedback from routers along the path via "message to big" ICMP packets in the reverse direction of the tunnel [RFC1191][RFC1981]. This technique is susceptible to the "black hole" phenomenon, in which the ICMP messages never return to the source due to policy-based filtering [RFC2923]. PLPMTUD requires a separate, direct control channel from the egress to the ingress that provides positive feedback; the direct channel is not blocked by policy filters and the positive feedback ensures fail-safe operation if feedback messages are lost [RFC4821].

4.3. Coordination Issues

IP tunnels interact with link layer signals and capabilities in a variety of ways. The following subsections address some key issues of these interactions. In general, they are again informed by treating a tunnel as any other link layer and considering the interactions between the IP layer and link layers [RFC3819].

4.3.1. Signaling

In the current Internet architecture, signaling goes upstream, either from routers along a path or from the destination, back toward the source. Such signals are typically contained in ICMP messages, but can involve other protocols such as RSVP, transport protocol signals (e.g., TCP RSTs), or multicast control or transport protocols.

A tunnel behaves like a link and acts like a link interface at the nodes where it is attached. As such, it can provide information that

enhances IP signaling (e.g., ICMP), but itself does not directly generate ICMP messages.

For tunnels, this means that there are two separate signaling paths. The outer network M nodes can each signal the source of the tunnel transit packets, Hsrc (Figure 14). Inside the tunnel, the inner network N nodes can signal the source of the tunnel link packets, the ingress I (Figure 15).

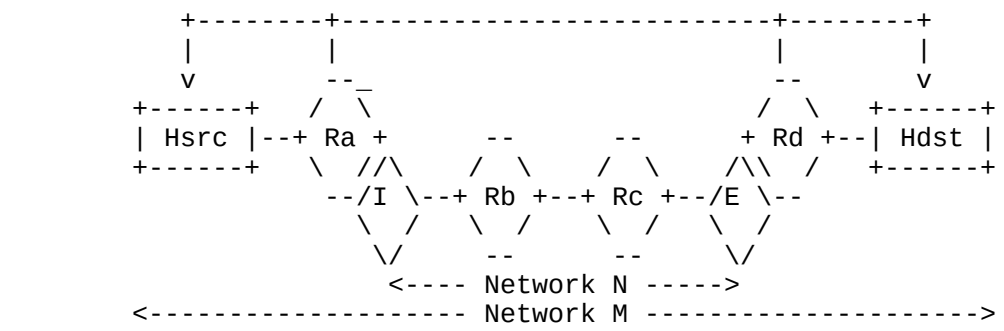


Figure 14 Signals outside the tunnel

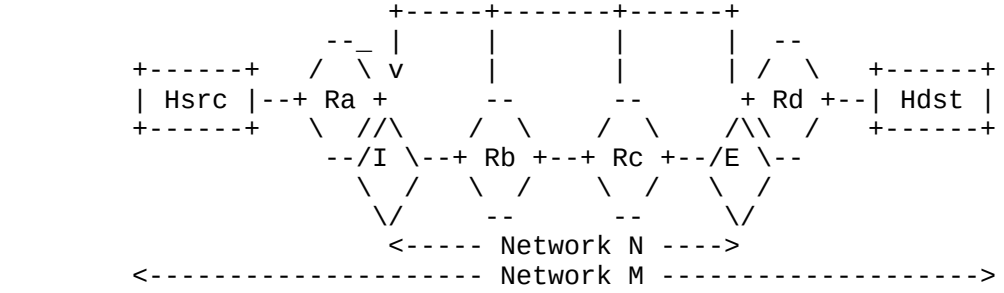


Figure 15 Signals inside the tunnel

These two signal paths are inherently distinct except where information is exchanged between the network interface of the tunnel (the ingress) and its attached node (Ra, in both figures).

It is always possible for a network interface to provide hints to its attached node (host or router), which can be used for optimization. In this case, when signals inside the tunnel indicate a change to the tunnel, the ingress (i.e., the tunnel network interface) can provide information to the router (Ra, in both figures), so that Ra can generate the appropriate signal in return to Hsrc. This relaying may

be difficult, because signals inside the tunnel may not return enough information to the ingress to support direct relaying to Hsrc.

In all cases, the tunnel ingress needs to determine how to relay the signals from inside the tunnel into signals back to the source. For some protocols this is either simple or impossible (such as for ICMP), for others, it can even be undefined (e.g., multicast). In some cases, the individual signals relayed from inside the tunnel may result in corresponding signals in the outside network, and in other cases they may just change state of the tunnel interface. In the latter case, the result may cause the router Ra to generate new ICMP errors when later messages arrive from Hsrc or other sources in the outer network.

The meaning of the relayed information must be carefully translated. An ICMP error within a tunnel indicates a failure of the path inside the tunnel to support an egress EMTU_S. It can be very difficult to convert that ICMP error into a corresponding ICMP message from the ingress node back to the transit packet source. The ICMP message may not contain enough of a packet prefix to extract the transit packet header sufficient to generate the appropriate ICMP message. The relationship between the egress EMTU_S and the transit packet may be indirect, e.g., the ingress node may be performing source fragmentation that should be adjusted instead of propagating the ICMP upstream.

Some messages have detailed specifications for relaying between the tunnel link packet and transit packet, including Explicit Congestion Notification (ECN [RFC6040]) and multicast (IGMP, e.g.).

4.3.2. Congestion

Tunnels carrying IP traffic (i.e., the focus of this document) need not react directly to congestion any more than would any other link layer [RFC8085]. IP transit packet traffic is already expected to be congestion controlled.

It is useful to relay network congestion notification between the tunnel link and the tunnel transit packets. Explicit congestion notification requires that ECN bits are copied from the tunnel transit packet to the tunnel link packet on encapsulation, as well as copied back at the egress based on a combination of the bits of the two headers [RFC6040]. This allows congestion notification within the tunnel to be interpreted as if it were on the direct path.

4.3.3. Multipoint Tunnels and Multicast

Multipoint tunnels are tunnels with more than two ingress/egress endpoints. Just as tunnels emulate links, multipoint tunnels emulate multipoint links, and can support multicast as a tunnel capability. Multipoint tunnels can be useful on their own, or may be used as part of more complex systems, e.g., LISP and TRILL configurations [RFC6830][RFC6325].

Multipoint tunnels require a support for egress determination, just as multipoint links do. This function is typically supported by ARP [RFC826] or ARP emulation (e.g., LAN Emulation, known as LANE [RFC2225]) for multipoint links. For multipoint tunnels, a similar mechanism is required for the same purpose - to determine the egress address for proper ingress encapsulation (e.g., LISP Map-Service [RFC6833]).

All multipoint systems - tunnels and links - might support different MTUs between each ingress/egress (or link entrance/exit) pair. In most cases, it is simpler to assume a uniform MTU throughout the multipoint system, e.g., the minimum MTU supported across all ingress/egress pairs. This applies to both the ingress EMTU_S and ingress EMTU_S (the latter determining the tunnel MTU).

A multipoint tunnel MUST have support for broadcast and multicast, in exactly the same way as this is already required for multipoint links [RFC3819]. Both modes can be supported either by a native mechanism inside the tunnel or by emulation using serial replication at the tunnel ingress (e.g., AMT [RFC7450]), in the same way that links may provide the same support either natively (e.g., via promiscuous or automatic replication in the link itself) or network interface emulation (e.g., as for non-broadcast multiaccess networks, i.e., NBMAs).

IGMP snooping enables IP multicast to be coupled with native link layer multicast support [RFC4541]. A similar technique may be relevant to couple transit packet multicast to tunnel link packet multicast, but the coupling of the protocols may be more complex because many tunnel link protocols rely on their own network N multicast control protocol, e.g., via PIM-SM [RFC6807][RFC7761].

4.3.4. Load Balancing

Load balancing can impact the way in which a tunnel operates. In particular, multipath routing inside the tunnel can impact some of the tunnel parameters to vary, both over time and for different transit packets. The use of multiple paths can be the result of MPLS

link aggregation groups (LAGs), equal-cost multipath routing (ECMP [RFC2991]), or other load balancing mechanisms. In some cases, the tunnel exists as the mechanism to support ECMP, as for GRE in UDP [RFC8086].

A tunnel may have multiple paths between the ingress and egress with different path MTU values, causing the ingress EMTU_S to vary [RFC7690]. Rather than track individual values, the EMTU_S can be set to the minimum of these different path MTU values.

IPv6 packets include a flow label to enable multipath routing to keep packets of a single flow following the same path. It is helpful to preserve the semantics of that flow label as an aggregate identifier inside the encapsulated link packets of a tunnel. This is achieved by hashing the transit IP addresses and flow label to generate a new flow label for use between the ingress and egress addresses [RFC6438]. It is not useful to simply copy the flow label from the transit packet into the link packet because of collisions that might arise if a label is used for flows between different transit packet addresses that traverse the same tunnel.

4.3.5. Recursive Tunnels

The rules described in this document already support tunnels over tunnels, sometimes known as "recursive" tunnels, in which IP is transited over IP either directly or via intermediate encapsulation (IP-UDP-IP, as in GUE [He16]).

There are known hazards to recursive tunneling, notably that the independence of the tunnel transit header and tunnel link header hop counts can result in a tunneling loop. Such looping can be avoided when using direct encapsulation (IP in IP) by use of a header option to track the encapsulation count and to limit that count [RFC2473]. This looping cannot be avoided when other protocols are used for tunneling, e.g., IP in UDP in IP, because the encapsulation count may not be visible where the recursion occurs.

5. Observations

The following subsections summarize the observations of this document and a summary of issues with existing tunnel protocol specifications. It also includes advice for tunnel protocol designers, implementers, and operators. It also includes

5.1. Summary of Recommendations

- Tunnel endpoints are network interfaces, tunnel are virtual links

- o ICMP messages MUST NOT be generated by the tunnel (as a link)
- o ICMP messages received by the ingress inside link change the link properties (they not generate transit-layer ICMP messages)
- o Link headers (hop, ID, options) are largely independent of arriving ID (with few exceptions based on translation, not direct copying, e.g., ECN and IPv6 flow IDs)

- o MTU values should treat the tunnel as any other link
  - o Require source ingress source fragmentation and egress reassembly at the tunnel link packet layer
  - o The tunnel MTU is the tunnel egress EMTU_S less headers, and not related at all to the ingress-egress MFS

- o Tunnels must obey core IP requirements
  - o Obey IPv4 DF=0 on arrival at a node (nodes MUST NOT fragment IPv4 packets where DF=0)
  - o Shut down an IP tunnel if the tunnel MTU falls below the required minimum

5.2. Impact on Existing Encapsulation Protocols

Many existing and proposed encapsulation protocols are inconsistent with the guidelines of this document. The following list summarizes only those inconsistencies, but omits places where a protocol is inconsistent solely by reference to another protocol.

[should this be inverted as a table of issues and a list of which RFCs have problems?]

- o IP in IP / mobile IP [RFC2003][RFC4459] - IPv4 in IPv4
  - o Sets link DF when transit DF=1 (fails without PLPMTUD)
  - o Drops at egress if hopcount = 0 (host-host tunnels fail)
  - o Drops based on transit source (same as router IP, matches egress), i.e., performs routing functions it should not

- o Ingress generates ICMP messages (based on relayed context), rather than using inner ICMP messages to set interface properties only
- o Treats tunnel MTU as tunnel path MTU, not tunnel egress MTU

- o IPv6 tunnels [RFC2473] -- IPv6 or IPv4 in IPv6
  - o Treats tunnel MTU as tunnel path MTU, not tunnel egress MTU
  - o Decrements transiting packet hopcount (by 1)
  - o Copies traffic class from tunnel link to tunnel transit header
  - o Ignores IPv4 DF=0 and fragments at that layer upon arrival
  - o Fails to retain soft ingress state based on inner ICMP messages affecting tunnel MTU
  - o Tunnel ingress issues ICMPs
  - o Fragments IPv4 over IPv6 fragments only if IPv4 DF=0 (misinterpreting the "can fragment the IPv4 packet" as permission to fragment at the IPv6 link header)
- o IPsec tunnel mode (IP in IPsec in IP) [RFC4301] -- IP in IPsec
  - o Uses security policy to set, clear, or copy DF (rather than generating it independently, which would also be more secure)
  - o Intertwines tunnel selection with security selection, rather than presenting tunnel as an interface and using existing forwarding (as with transport mode over IP-in-IP [RFC3884])
- o GRE (IP in GRE in IP or IP in GRE in UDP in IP) [RFC2784][RFC7588][RFC7676][RFC8086]
  - o Treats tunnel MTU as tunnel path MTU, not tunnel egress MTU
  - o Requires ingress to generate ICMP errors
  - o Copies IPv4 DF to outer IPv4 DF
  - o Violates IPv6 MTU requirements when using IPv6 encapsulation
- o LISP [RFC6830]

- Treats tunnel MTU as tunnel path MTU, not tunnel egress MTU
- Requires ingress to generate ICMP errors
- Copies inner hop limit to outer

- L2TP [RFC3931]
  - Treats tunnel MTU as tunnel path MTU, not tunnel egress MTU
  - Requires ingress to generate ICMP errors
- PWE [RFC3985]
  - Treats tunnel MTU as tunnel path MTU, not tunnel egress MTU
  - Requires ingress to generate ICMP errors
- GUE (Generic UDP encapsulation) [He16] - IP (et. al) in UDP in IP
  - Allows inner encapsulation fragmentation
- Geneve [RFC7364][Gr16] - IP (et al.) in Geneve in UDP in IP
  - Treats tunnel MTU as tunnel path MTU, not tunnel egress MTU
- SEAL/AERO [RFC5320][Te16] - IP in SEAL/AERO in IP
  - Some issues with SEAL (MTU, ICMP), corrected in AERO
- RTG DT encapsulations [No16]
  - Assumes fragmentation can be avoided completely
  - Allows encapsulation protocols that lack fragmentation
  - Relies on ICMP PTB to correct for tunnel path MTU
- No known issues
  - L2VPN (framework for L2 virtualization) [RFC4664]
  - L3VPN (framework for L3 virtualization) [RFC4176]
  - MPLS (IP in MPLS) [RFC3031]
  - TRILL (Ethernet in Ethernet) [RFC5556][RFC6325]

5.3. Tunnel Protocol Designers

[To be completed]

Recursive tunneling + minimum MTU = frag/reassembly is inevitable, at least to be able to split/join two fragments

Account for egress MTU/path MTU differences.

Include a stronger checksum.

Ensure the egress MTU is always larger than the path MTU.

Ensure that the egress reassembly can keep up with line rate OR design PLPMTUD into the tunneling protocol.

5.3.1. For Future Standards

[To be completed]

Larger IPv4 MTU (2K? or just 2x path MTU?) for reassembly

Always include frag support for at least two frags; do NOT try to deprecate fragmentation.

Limit encapsulation option use/space.

Augment ICMP to have two separate messages: PTB vs P-bigger-than-optimal

Include MTU as part of BGP as a hint - SB

Hazards of multi-MTU draft-van-beijnum-multi-mtu-04

5.3.2. Diagnostics

[To be completed]

Some current implementations include diagnostics to support monitoring the impact of tunneling, especially the impact on fragmentation and reassembly resources, the status of path MTU discovery, etc.

>> Because a tunnel ingress/egress is a network interface, it SHOULD have similar resources as any other network interface. This includes resources for packet processing as well as monitoring.

5.4. Tunnel Implementers

[To be completed]

Detect when the egress MTU is exceeded.

Detect when the egress MTU drops below the required minimum and shut down the tunnel if that happens - configuring the tunnel down and issuing a hard error may be the only way to detect this anomaly, and it’s sufficiently important that the tunnel SHOULD be disabled. This is always better than blindly assuming the tunnel has been deployed correctly, i.e., that the solution has been engineered.

Do NOT decrement the TTL as part of being a tunnel. It’s always already OK for a router to decrement the TTL based on different next-hop routers, but TTL is a property of a router not a link.

5.5. Tunnel Operators

[To be completed]

Keep the difference between "enforced by operators" vs. "enforced by active protocol mechanism" in mind. It’s fine to assume something the tunnel cannot or does not test, as long as you KNOW you can assume it. When the assumption is wrong, it will NOT be signaled by the tunnel. Do NOT decrement the TTL as part of being a tunnel. It’s always already OK for a router to decrement the TTL based on different next-hop routers, but TTL is a property of a router not a link.

Consider the circuit breakers doc to provide diagnostics and last-resort control to avoid overload for non-reactive traffic (see Gorry’s RFC-to-be)

Do NOT decrement the TTL as part of being a tunnel. It’s always already OK for a router to decrement the TTL based on different next-hop routers, but TTL is a property of a router not a link.

>>>> PLPMTUD can give multiple conflicting PMTU values during ECMP or LAG if PMTU is cached per endpoint pair rather than per flow -- but so can PMTUD! This is another reason why ICMP should never drive up the effective MTU (if aggregate, treat as the minimum of received messages over an interval).

6. Security Considerations

Tunnels may introduce vulnerabilities or add to the potential for receiver overload and thus DOS attacks. These issues are primarily related to the fact that a tunnel is a link that traverses a network path and to fragmentation and reassembly. ICMP signal translation introduces a new security issue and must be done with care. ICMP generation at the router or host attached to a tunnel is already covered by existing requirements (e.g., should be throttled).

Tunnels traverse multiple hops of a network path from ingress to egress. Traffic along such tunnels may be susceptible to on-path and off-path attacks, including fragment injection, reassembly buffer overload, and ICMP attacks. Some of these attacks may not be as visible to the endpoints of the architecture into which tunnels are deployed and these attacks may thus be more difficult to detect.

Fragmentation at routers or hosts attached to tunnels may place an undue burden on receivers where traffic is not sufficiently diffuse, because tunnels may induce source fragmentation at hosts and path fragmentation (for IPv4 DF=0) more for tunnels than for other links. Care should be taken to avoid this situation, notably by ensuring that tunnel MTUs are not significantly different from other link MTUs.

Tunnel ingresses emitting IP datagrams MUST obey all existing IP requirements, such as the uniqueness of the IP ID field. Failure to either limit encapsulation traffic, or use additional ingress/egress IP addresses, can result in high speed traffic fragments being incorrectly reassembled.

Tunnels are susceptible to attacks at both the inner and outer network layers. The tunnel ingress/egress endpoints appear as network interfaces in the outer network, and are as susceptible as any other network interface. This includes vulnerability to fragmentation reassembly overload, traffic overload, and spoofed ICMP messages that misreport the state of those interfaces. Similarly, the ingress/egress appear as hosts to the path traversed by the tunnel, and thus are as susceptible as any other host to attacks as well.

[management?]

[Access control?]

describe relationship to [RFC6169] - JT (as per INTAREA meeting notes, don’t cover Teredo-specific issues in RFC6169, but include generic issues here)

7. IANA Considerations

This document has no IANA considerations.

The RFC Editor should remove this section prior to publication.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[are there others? 3819? ECN? Flow label issues?]

8.2. Informative References

[Cl88] Clark, D., "The design philosophy of the DARPA internet protocols," Proc. Sigcomm 1988, p.106-114, 1988.

[Er94] Eriksson, H., "MBone: The Multicast Backbone," Communications of the ACM, Aug. 1994, pp.54-60.

[Gr16] Gross, J. (Ed.), I. Ganga (Ed.), T. Sridhar (Ed.), "Geneve: Generic Network Virtualization Encapsulation," draft-ietf-nvo3-geneve-03, Sep. 2016.

[He16] Herbert, T., L. Yong, O. Zia, "Generic UDP Encapsulation," draft-ietf-nvo3-gue-05, Oct. 2016.

[Ke95] Kent, S., J. Mogul, "Fragmentation considered harmful," ACM Sigcomm Computer Communication Review (CCR), V25 N1, Jan. 1995, pp. 75-87.

[No16] Nordmark, E. (Ed.), A. Tian, J. Gross, J. Hudson, L. Kreeger, P. Garg, P. Thaler, T. Herbert, "Encapsulation Considerations," draft-ietf-rtgwg-dt-encap-02, Oct. 2016.

[RFC5] Rulifson, J, "Decode Encode Language (DEL)," RFC 5, June 1969.

[RFC768] Postel, J, "User Datagram Protocol," RFC 768, Aug. 1980

[RFC791] Postel, J., "Internet Protocol," RFC 791 / STD 5, September 1981.

[RFC792] Postel, J., "Internet Control Message Protocol," RFC 792, Sep. 981.

[RFC793] Postel, J, "Transmission Control Protocol," RFC 793, Sept. 1981.

[RFC826] Plummer, D., "An Ethernet Address Resolution Protocol -- or -- Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware," RFC 826, Nov. 1982.

[RFC1075] Waitzman, D., C. Partridge, S. Deering, "Distance Vector Multicast Routing Protocol," RFC 1075, Nov. 1988.

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers," RFC 1122 / STD 3, October 1989.

[RFC1191] Mogul, J., S. Deering, "Path MTU discovery," RFC 1191, November 1990.

[RFC1812] Baker, F., "Requirements for IP Version 4 Routers," RFC 1812, June 1995.

[RFC1853] Simpson, W., "IP in IP Tunneling," RFC 1853, Oct. 1995.

[RFC1981] McCann, J., S. Deering, J. Mogul, "Path MTU Discovery for IP version 6," RFC 1981, Aug. 1996.

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

[RFC2225] Laubach, M., J. Halpern, "Classical IP and ARP over ATM," RFC 2225, Apr. 1998.

[RFC2460] Deering, S., R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," RFC 2460, Dec. 1998.

[RFC2473] Conta, A., "Generic Packet Tunneling in IPv6 Specification," RFC 2473, Dec. 1998.

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

[RFC2923] Lahey, K., "TCP Problems with Path MTU Discovery," RFC 2923, September 2000.

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

[RFC2991] Thaler, D., C. Hopps, "Multipath Issues in Unicast and Multicast Next-Hop Selection," RFC 2991, Nov. 2000.

[RFC2473] Conta, A., S. Deering, "Generic Packet Tunneling in IPv6 Specification," RFC 2473, Dec. 1998.

[RFC2546] Durand, A., B. Buclin, "6bone Routing Practice," RFC 2540, Mar. 1999.

[RFC3031] Rosen, E., A. Viswanathan, R. Callon, "Multiprotocol Label Switching Architecture", RFC 3031, January 2001.

[RFC3819] Karn, P., Ed., C. Bormann, G. Fairhurst, D. Grossman, R. Ludwig, J. Mahdavi, G. Montenegro, J. Touch, L. Wood, "Advice for Internet Subnetwork Designers," RFC 3819 / BCP 89, July 2004.

[RFC3884] Touch, J., L. Eggert, Y. Wang, "Use of IPsec Transport Mode for Dynamic Routing," RFC 3884, September 2004.

[RFC3931] Lau, J., Ed., M. Townsley, Ed., I. Goyret, Ed., "Layer Two Tunneling Protocol - Version 3 (L2TPv3)," RFC 3931, March 2005.

[RFC3985] Bryant, S., P. Pate (Eds.), "Pseudo Wire Emulation Edge-to-Edge (PWE3) Architecture", RFC 3985, March 2005.

[RFC4176] El Mghazli, Y., Ed., T. Nadeau, M. Boucadair, K. Chan, A. Gonguet, "Framework for Layer 3 Virtual Private Networks (L3VPN) Operations and Management," RFC 4176, October 2005.

[RFC4301] Kent, S., and K. Seo, "Security Architecture for the Internet Protocol," RFC 4301, December 2005.

[RFC4340] Kohler, E., M. Handley, S. Floyd, "Datagram Congestion Control Protocol (DCCP)," RFC 4340, Mar. 2006.

[RFC4443] Conta, A., S. Deering, M. Gupta (Ed.), "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification," RFC 4443, Mar. 2006.

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

[RFC4541] Christensen, M., K. Kimball, F. Solensky, "Considerations for Internet Group Management Protocol (IGMP) and Multicast Listener Discovery (MLD) Snooping Switches," RFC 4541, May 2006.

[RFC4664] Andersson, L., Ed., E. Rosen, Ed., "Framework for Layer 2 Virtual Private Networks (L2VPNs)," RFC 4664, September 2006.

[RFC4821] Mathis, M., J. Heffner, "Packetization Layer Path MTU Discovery," RFC 4821, March 2007.

[RFC4861] Narten, T., E. Nordmark, W. Simpson, H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)," RFC 4861, Sept. 2007.

[RFC4960] Stewart, R. (Ed.), "Stream Control Transmission Protocol," RFC 4960, Sep. 2007.

[RFC4963] Heffner, J., M. Mathis, B. Chandler, "IPv4 Reassembly Errors at High Data Rates," RFC 4963, July 2007.

[RFC5320] Templin, F., Ed., "The Subnetwork Encapsulation and Adaptation Layer (SEAL)," RFC 5320, Feb. 2010.

[RFC5556] Touch, J., R. Perlman, "Transparently Interconnecting Lots of Links (TRILL): Problem and Applicability Statement," RFC 5556, May 2009.

[RFC5944] Perkins, C., Ed., "IP Mobility Support for IPv4, Revised" RFC 5944, Nov. 2010.

[RFC6040] Briscoe, B., "Tunneling of Explicit Congestion Notification," RFC 6040, Nov. 2010.

[RFC6169] Krishnan, S., D. Thaler, J. Hoagland, "Security Concerns With IP Tunneling," RFC 6169, Apr. 2011.

[RFC6325] Perlman, R., D. Eastlake, D. Dutt, S. Gai, A. Ghanwani, "Routing Bridges (RBridges): Base Protocol Specification," RFC 6325, July 2011.

[RFC6434] Jankiewicz, E., J. Loughney, T. Narten, "IPv6 Node Requirements," RFC 6434, Dec. 2011.

[RFC6438] Carpenter, B., S. Amante, "Using the IPv6 Flow Label for Equal Cost Multipath Routing and Link Aggregation in Tunnels," RFC 6438, Nov. 2011.

[RFC6807] Farinacci, D., G. Shepherd, S. Venaas, Y. Cai, "Population Count Extensions to Protocol Independent Multicast (PIM)," RFC 6807, Dec. 2012.

[RFC6830] Farinacci, D., V. Fuller, D. Meyer, D. Lewis, "The Locator/ID Separation Protocol," RFC 6830, Jan. 2013.

[RFC6833] Fuller, V., D. Farinacci, "Locator/ID Separation Protocol (LISP) Map-Server Interface," RFC 6833, Jan. 2013.

[RFC6864] Touch, J., "Updated Specification of the IPv4 ID Field," Proposed Standard, RFC 6864, Feb. 2013.

[RFC6935] Eubanks, M., P. Chimento, M. Westerlund, "IPv6 and UDP Checksums for Tunneled Packets," RFC 6935, Apr. 2013.

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

[RFC6946] Gont, F., "Processing of IPv6 "Atomic" Fragments," RFC 6946, May 2013.

[RFC7364] Narten, T., Gray, E., Black, D., Fang, L., Kreeger, L., M. Napierala, "Problem Statement: Overlays for Network Virtualization", RFC 7364, Oct. 2014.

[RFC7450] Bumgardner, G., "Automatic Multicast Tunneling," RFC 7450, Feb. 2015.

[RFC7510] Xu, X., N. Sheth, L. Yong, R. Callon, D. Black, "Encapsulating MPLS in UDP," RFC 7510, April 2015.

[RFC7588] Bonica, R., C. Pignataro, J. Touch, "A Widely-Deployed Solution to the Generic Routing Encapsulation Fragmentation Problem," RFC 7588, July 2015.

[RFC7676] Pignataro, C., R. Bonica, S. Krishnan, "IPv6 Support for Generic Routing Encapsulation (GRE)," RFC 7676, Oct 2015.

[RFC7690] Byerly, M., M. Hite, J. Jaeggli, "Close Encounters of the ICMP Type 2 Kind (Near Misses with ICMPv6 Packet Too Big (PTB))," RFC 7690, Jan. 2016.

[RFC7761] Fenner, B., M. Handley, H. Holbrook, I. Kouvelas, R. Parekh, Z. Zhang, L. Zheng, "Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised)," RFC 7761, Mar. 2016.

[RFC8085] Eggert, L., G. Fairhurst, G. Shepherd, "Unicast UDP Usage Guidelines," RFC 8085, Oct. 2015.

[RFC8086] Yong, L. (Ed.), E. Crabbe, X. Xu, T. Herbert, "GRE-in-UDP Encapsulation," RFC 8086, Feb. 2017.

[Sa84] Saltzer, J., D. Reed, D. Clark, "End-to-end arguments in system design," ACM Trans. on Computing Systems, Nov. 1984.

[Te16] Templin, F., "Asymmetric Extended Route Optimization," draft-templin-aerolink-74, Nov. 2016.

[To01] Touch, J., "Dynamic Internet Overlay Deployment and Management Using the X-Bone," Computer Networks, July 2001, pp. 117-135.

[To03] Touch, J., Y. Wang, L. Eggert, G. Finn, "Virtual Internet Architecture," USC/ISI Tech. Report ISI-TR-570, Aug. 2003.

[To16] Touch, J., "Middleboxes Models Compatible with the Internet," USC/ISI Tech. Report ISI-TR-711, Oct. 2016.

[To98] Touch, J., S. Hotz, "The X-Bone," Proc. Globecom Third Global Internet Mini-Conference, Nov. 1998.

[Zi80] Zimmermann, H., "OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection," IEEE Trans. on Comm., Apr. 1980.

9. Acknowledgments

This document originated as the result of numerous discussions among the authors, Jari Arkko, Stuart Bryant, Lars Eggert, Ted Faber, Gorry Fairhurst, Dino Farinacci, Matt Mathis, and Fred Templin. It benefitted substantially from detailed feedback from Toerless Eckert, Vincent Roca, and Lucy Yong, as well as other members of the Internet Area Working Group.

This work is partly supported by USC/ISI’s Postel Center.

This document was prepared using 2-Word-v2.0.template.dot.

Authors' Addresses

Joe Touch
USC/ISI
4676 Admiralty Way
Marina del Rey, CA 90292-6695
U.S.A.

Phone: +1 (310) 448-9151
Email: touch@isi.edu

W. Mark Townsley
Cisco
L'Atlantis, 11, Rue Camille Desmoulins
Issy Les Moulineaux, ILE DE FRANCE 92782

Email: townsley@cisco.com

APPENDIX A: Fragmentation efficiency

A.1. Selecting fragment sizes

There are different ways to fragment a packet. Consider a network with a PMTU as shown in Figure 16, where packets are encapsulated over the same network layer as they arrive on (e.g., IP in IP). If a packet as large as the PMTU arrives, it must be fragmented to accommodate the additional header.

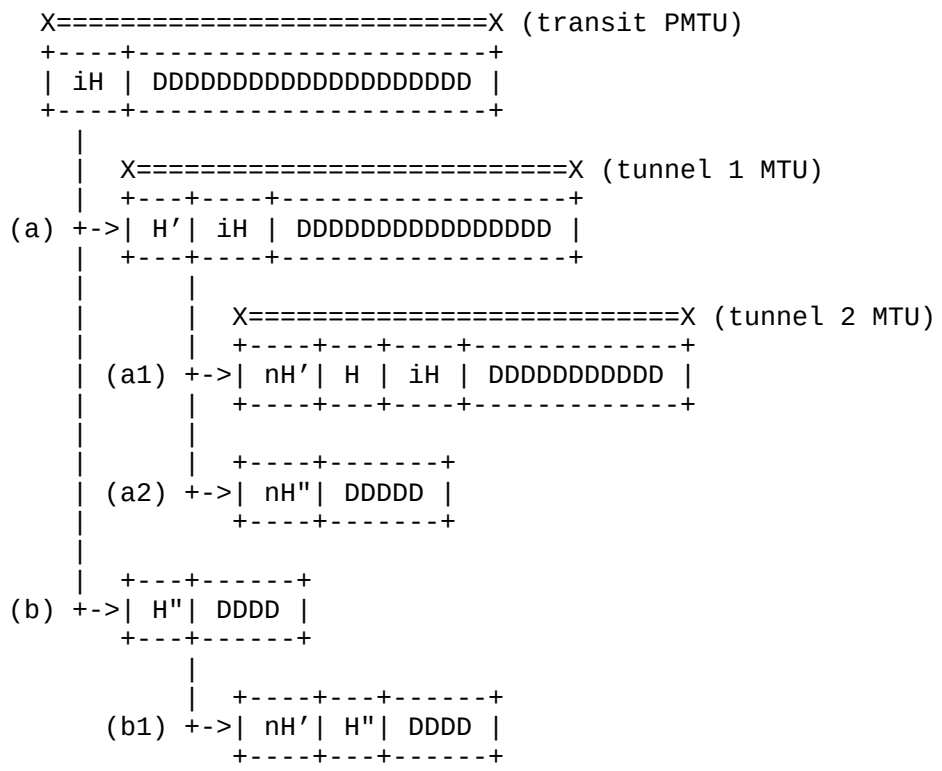


Figure 16 Fragmenting via maximum fit

Figure 16 shows this process using "maximum fit", assuming outer fragmentation as an example (the situation is the same for inner fragmentation, but the headers that are affected differ). In maximum fit, the arriving packet is split into (a) and (b), where (a) is the size of the first tunnel, i.e., the tunnel 1 MTU (the maximum that fits over the first tunnel). However, this tunnel then traverses over another tunnel (number 2), whose impact the first tunnel ingress has not accommodated. The packet (a) arrives at the second tunnel

ingress, and needs to be encapsulated again, but it needs to be fragmented as well to fit into the tunnel 2 MTU, into (a1) and (a2). In this case, packet (b) arrives at the second tunnel ingress and is encapsulated into (b1) without fragmentation, because it is already below the tunnel 2 MTU size.

In Figure 17, the fragmentation is done using "even split", i.e., by splitting the original packet into two roughly equal-sized components, (c) and (d). Note that (d) contains more packet data, because (c) includes the original packet header because this is an example of outer fragmentation. The packets (c) and (d) arrive at the second tunnel encapsulator, and are encapsulated again; this time, neither packet exceeds the tunnel 2 MTU, and neither requires further fragmentation.

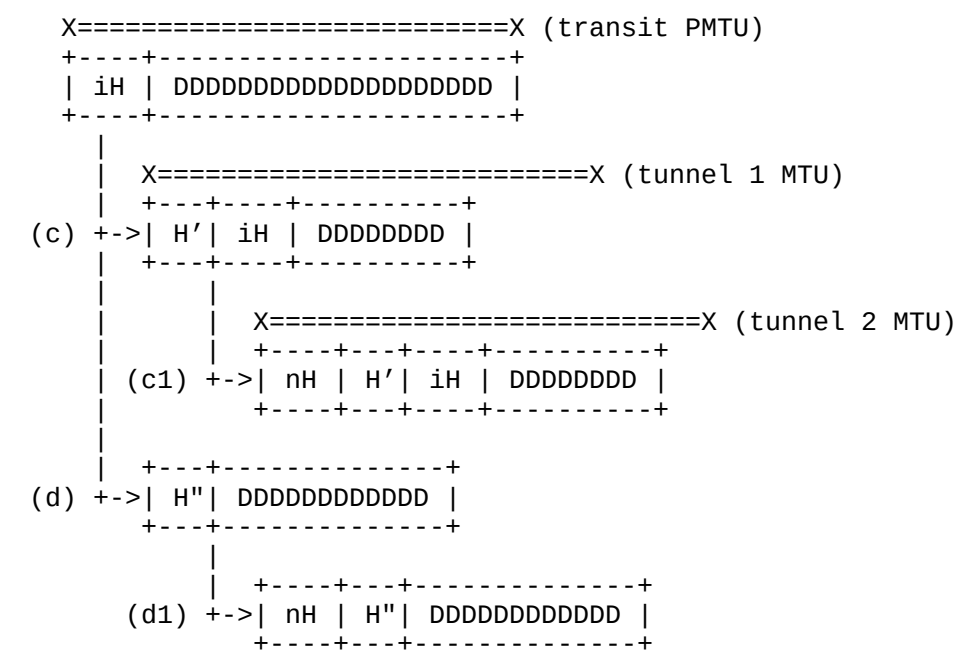


Figure 17 Fragmenting via "even split"

A.2. Packing

Encapsulating individual packets to traverse a tunnel can be inefficient, especially where headers are large relative to the packets being carried. In that case, it can be more efficient to encapsulate many small packets in a single, larger tunnel payload.

This technique, similar to the effect of packet bursting in Gigabit Ethernet (regardless of whether they're encoded using L2 symbols as delineators), reduces the overhead of the encapsulation headers (Figure 18). It reduces the work of header addition and removal at the tunnel endpoints, but increases other work involving the packing and unpacking of the component packets carried.

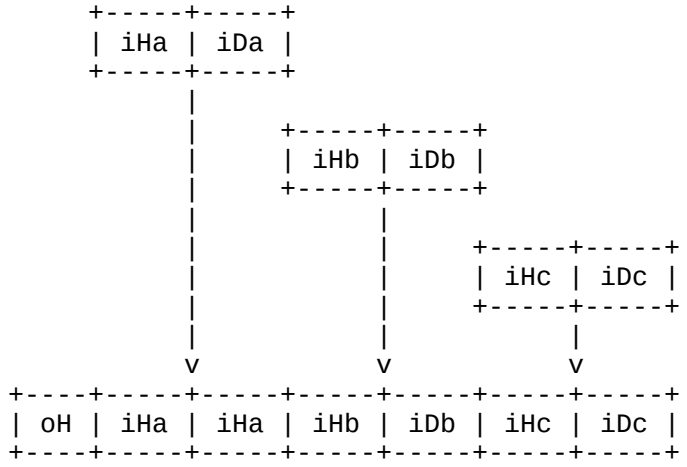


Figure 18 Packing packets into a tunnel

intarea Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 7, 2017

L. Li
Y. Cui
C. Liu
J. Wu
Tsinghua University
F. Baker

J. Palet Martinez
Consulintel, S.L.
March 6, 2017

DHCPv6 Options for Discovery NAT64 Prefixes
draft-li-intarea-nat64-prefix-dhcp-option-00

Abstract

Several IPv6 transition mechanisms require the usage of stateless or stateful translators (commonly named as NAT64) able to allow IP/ICMP communication between IPv4 and IPv6 networks.

Those translators are using either a default well-known prefix, and/or one or several additional network specific prefixes, which need to be configured into the nodes willing to use the translator. Different translators will likely have different IPv6 prefixes, to attract traffic to the correct translator. Thus, an automatic translator prefix discovery method is necessary.

This document defines a DHCPv6-based method to inform DHCPv6 clients the set of IPv6 and IPv4 prefixes it serves. This DHCPv6 option can be used by several transition mechanisms such as SIIT, 464XLAT, EAM.

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 September 7, 2017.

Table of Contents

1. Introduction

Stateless IP/ICMP Translation (SIIT) [RFC7915] describes the basic translation mechanism (NAT64), which is actually used as the base for most of the related translation protocols.

Stateful NAT64 [RFC6146] describes how to allow IPv6-only clients to contact IPv4 servers using unicast UDP, TCP or ICMP.

464XLAT [RFC6877] describes an IPv4-over-IPv6 solution as one technique for IPv4 service extension and encouragement of IPv6 deployment. The 464XLAT architecture uses IPv4/IPv6 translation, described in [RFC6144], and standardized in [RFC6052], [RFC7915], and [RFC6146]. It encourages the IPv6 transition by making IPv4 service reachable across IPv6-only networks and providing IPv6 and IPv4 connectivity to single-stack IPv4 or IPv6 servers and peers. In the

464XLAT architecture, the CLAT (customer-side NAT46 translator) must determine which of potentially several PLAT (provider-side NAT64 translator) IPv6 prefix to use in order to send a packet to the PLAT with connectivity to its destination.

[RFC7050] describes a mechanism to learn the PLAT-side IPv6 prefix for protocol translation by DNS64 [RFC6147]. Although it supports multiple PLAT-side prefix by responding with multiple AAAA records to a DNS64 query, it does not support mapping IPv4 prefixes to IPv6 prefix, which would be required, for example, if one PLAT has connectivity to the general Internet following a default route, another has connectivity to a BGP peer, and a third has connectivity to a network using private addressing [RFC1918]. Therefore, in the scenario with multiple PLATs, [RFC7050] does not directly support destination-based IPv4 routing among PLATs; instead, the DNS64 database must contain equivalent information. It also requires the additional deployment of DNS64 service in customer-side networks, which is not required in 464XLAT deployment.

464XLAT is in fact, a usage case of Stateful NAT64.

Explicit Address Mappings for Stateless IP/ICMP Translation [RFC7757] extends SIIT with an Explicit Address Mapping (EAM) algorithm to facilitate stateless IP/ICMP translation between arbitrary (non-IPv4-translatable) IPv6 endpoints and IPv4.

This document proposes a method for the translator (NAT64) IPv6 prefix discovery based on DHCPv6, which is widely deployed and supported in customer networks. It defines two new DHCPv6 options for use by a DHCPv6 client to discover the translator IPv6 prefix(es). Also, the proposed mechanism can deal with the scenario with multiple independent DNS64 databases supporting separate translators.

2. Requirements Language

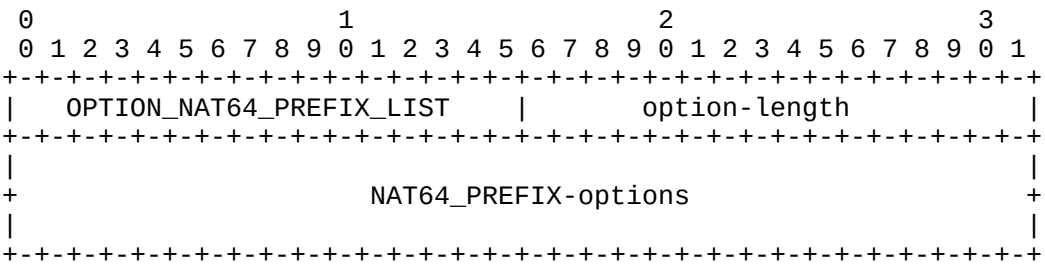
The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. New DHCPv6 Option

3.1. NAT64 Prefix List Option Format

The NAT Prefix List Option is a container for NAT64 Prefix Option(s). A NAT64 Prefix List Option MAY contain multiple NAT64 Prefix Options.

The format of the NAT64 Prefix List Option is:

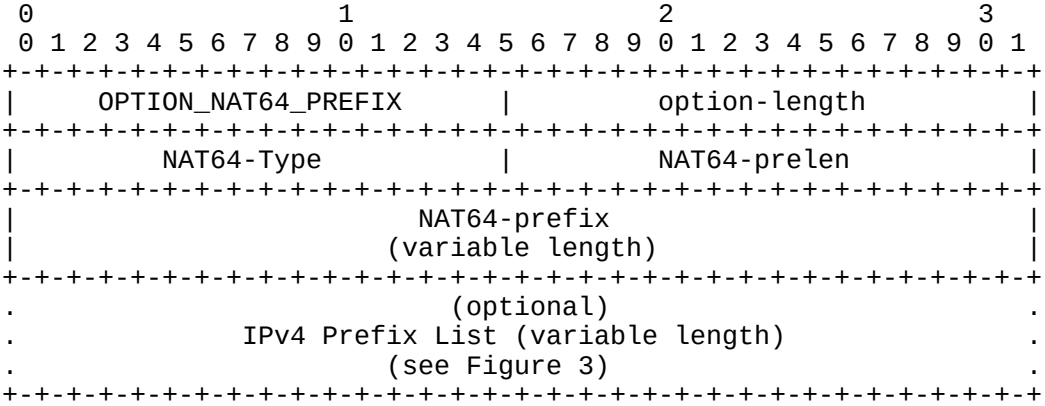


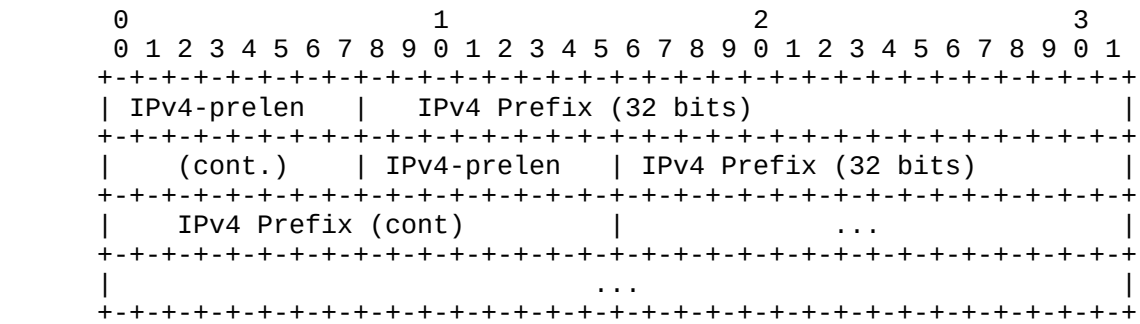
- option-code: OPTION_NAT64_PREFIX_LIST (TBA1)
- option-length: length of NAT64_PREFIX-options, specified in octets.
- NAT64_PREFIX-options: one or more OPTION_NAT64_PREFIX options.

3.2. NAT64 Prefix Option Format

The NAT64 Prefix Option is encapsulated in the NAT64 Prefix List Option. This option allows the mapping of destination IPv4 address ranges (contained in the IPv4 Prefix List) to a NAT64 IPv6 prefix. If there is more than one such prefix, each prefix comes in its own option, with its associated IPv4 prefix list. In this way, the DHCPv6 client can select the NAT64 with the corresponding destination IPv4 address.

The format of the NAT64 Prefix Option is:





- option-code: OPTION_NAT64_PREFIX (TBA2)

- type-field: NAT64-Type (TBA3)

- option-length: 1 + length of NAT64-prefix + length of IPv4 Prefix List, specified in octets.

- NAT64-prelen: length of NAT64-prefix.

- NAT64-prefix: The NAT64 IPv6 prefix that the DHCPv6 client use for IPv6 address synthesis.

- IPv4 Prefix List: This is an optional field. The format of the IPv4 Prefix List is shown in Figure 3. It is a list of zero or more IPv4 Prefixes. Each entry is formed by IPv4-prelen and IPv4 Prefix. The total length of the field is 5*number of IPv4 prefixes.

- IPv4-prelen: the length of the IPv4 Prefix.

- IPv4 Prefix: the destination-based IPv4 Prefix. The length is 4 octets.

4. Client Behavior

The client requests the OPTION_NAT64_PREFIX_LIST option using the Option Request option (ORO) in every Solicit, Request, Renew, Rebind, and Information-request message. The NAT64-Type field defines the mechanism being used. If the DHCPv6 server includes the OPTION_NAT64_PREFIX_LIST option in its response, the DHCPv6 client may use the contained NAT64-prefix to translate the destination IPv4 address into the destination IPv6 address.

When receiving the OPTION_NAT64_PREFIX option with IPv4 Prefix List, the DHCPv6 client MUST record the received IPv6 prefix and the corresponding IPv4 prefixes in IPv4 Prefix List. When receiving the

OPTION_NAT64_PREFIX option without IPv4 Prefix List, the DHCPv6 client MUST treat the IPv6 prefix and the default IPv4 prefix 0.0.0.0/0 as one of the records.

If the DHCPv6 client loses contact with the DHCPv6 server, the DHCPv6 client SHOULD clear the prefix(es) it learned from the DHCPv6 server.

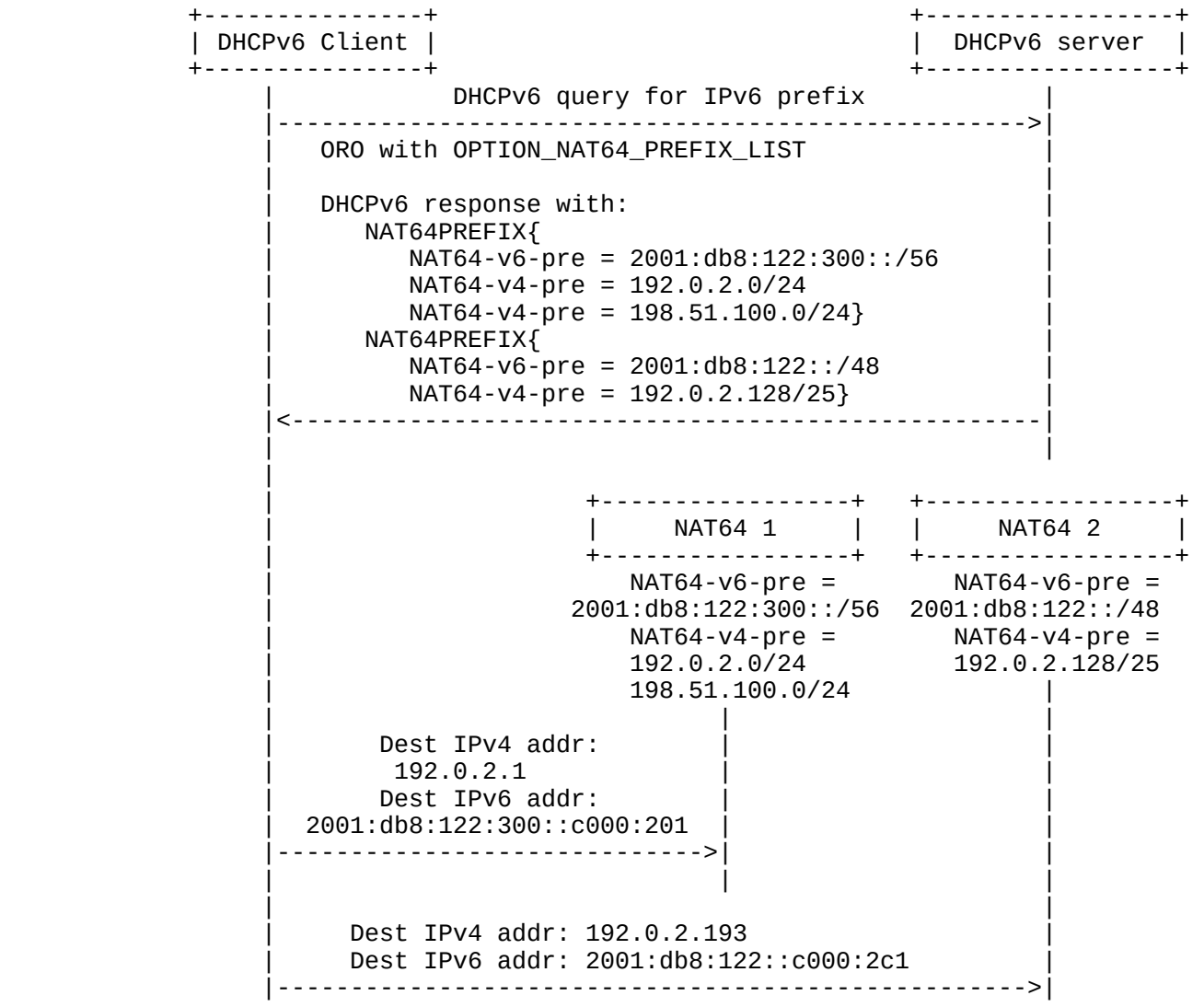
When translating the destination IPv4 address into the destination IPv6 address, DHCPv6 client MUST search an IPv4 routing database using the longest-match-first rule and select the IPv6 prefix offering that IPv4 prefix.

5. Message Flow Illustration

The figure below shows an example of message flow for a Client learning IPv6 prefixes using DHCPv6.

In this example, two IPv6 prefixes are provided by the DHCPv6 server. The first IPv6 prefix is 2001:db8:122:300::/56, the corresponding IPv4 prefixes are 192.0.2.0/24 and 198.51.100.0/24. The second IPv6 prefix is 2001:db8:122::/48, the corresponding IPv4 prefix is 192.0.2.128/25.

When the DHCPv6 client receives the packet with destination IPv4 address 192.0.2.1, according to the rule of longest prefix match, the NAT64 with IPv6 prefix 2001:db8:122::/48 is chosen. In the same way, the NAT64 with IPv6 prefix 2001:db8:122::/48 is chosen.



6. Security Considerations

Considerations for security in this type of environment are primarily around the operation of the DHCPv6 protocol and the databases it uses.

In the DHCPv6 server, should the database be compromised, it will deliver incorrect data to its DHCPv6 clients. In the DHCPv6 client, should its database be compromised by attack or polluted by an incorrect DHCPv6 server database, it will route data incorrectly. In

both cases, the security of the systems and their databases in an operational matter, not managed by protocol.

However, the operation of the DHCPv6 protocol itself is also required to be correct - the server and its clients must recognize valid requests and reject invalid ones. Therefore, DHCPv6 exchanges MUST be secured as described in [RFC3315].

7. IANA Considerations

We request that IANA allocate two DHCPv6 option codes for use by OPTION_V6_PLATPREFIX_LIST and OPTION_V6_PLATPREFIX from the "Option Codes" table. Similarly, a request to IANA for assigning the NAT64-Type field codes. The following initial values are assigned in this document (values are 16-bit unsigned intergers).

Name	Value	RFC
Unspecified	0x00	RFC6052
SIIT	0x01	RFC7915
Stateful NAT64	0x02	RFC6146
EAM-SIIT	0x03	RFC7757

8. Acknowledgements

The authors will like to recognize the inputs from Tore Anderson in a previous version of this work.

9. References

9.1. 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>.

[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>.

[RFC6877] Mawatari, M., Kawashima, M., and C. Byrne, "464XLAT: Combination of Stateful and Stateless Translation", RFC 6877, DOI 10.17487/RFC6877, April 2013, <http://www.rfc-editor.org/info/rfc6877>.

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>.

[RFC6052] Bao, C., Huitema, C., Bagnulo, M., Boucadair, M., and X. Li, "IPv6 Addressing of IPv4/IPv6 Translators", RFC 6052, DOI 10.17487/RFC6052, October 2010, <http://www.rfc-editor.org/info/rfc6052>.

[RFC6144] Baker, F., Li, X., Bao, C., and K. Yin, "Framework for IPv4/IPv6 Translation", RFC 6144, DOI 10.17487/RFC6144, April 2011, <http://www.rfc-editor.org/info/rfc6144>.

[RFC6146] Bagnulo, M., Matthews, P., and I. van Beijnum, "Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers", RFC 6146, DOI 10.17487/RFC6146, April 2011, <http://www.rfc-editor.org/info/rfc6146>.

[RFC6147] Bagnulo, M., Sullivan, A., Matthews, P., and I. van Beijnum, "DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers", RFC 6147, DOI 10.17487/RFC6147, April 2011, <http://www.rfc-editor.org/info/rfc6147>.

[RFC7050] Savolainen, T., Korhonen, J., and D. Wing, "Discovery of the IPv6 Prefix Used for IPv6 Address Synthesis", RFC 7050, DOI 10.17487/RFC7050, November 2013, <http://www.rfc-editor.org/info/rfc7050>.

[RFC7757] Anderson, T. and A. Leiva Popper, "Explicit Address Mappings for Stateless IP/ICMP Translation", RFC 7757, DOI 10.17487/RFC7757, February 2016, <http://www.rfc-editor.org/info/rfc7757>.

[RFC7915] Bao, C., Li, X., Baker, F., Anderson, T., and F. Gont, "IP/ICMP Translation Algorithm", RFC 7915, DOI 10.17487/RFC7915, June 2016, <http://www.rfc-editor.org/info/rfc7915>.

Authors' Addresses

Lishan Li
Tsinghua University
Beijing 100084
P.R.China

Phone: +86-15201441862
Email: lilishan9248@126.com

Yong Cui
Tsinghua University
Beijing 100084
P.R.China

Phone: +86-10-6260-3059
Email: yong@csnet1.cs.tsinghua.edu.cn

Cong Liu
Tsinghua University
Beijing 100084
P.R.China

Phone: +86-10-6278-5822
Email: gnocuil@gmail.com

Jianping Wu
Tsinghua University
Beijing 100084
P.R.China

Phone: +86-10-6278-5983
Email: jianping@cernet.edu.cn

Fred Baker
Goleta, CA 93117
United States

Email: fredbaker.ietf@gmail.com

Jordi Palet Martinez
Consulintel, S.L.
La Navata - Galapagar 28420
Spain

Email: jordi.palet@consulintel.es

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.

Table of Contents

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 use to attach nodes plus an inter-bridge port:

- 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.
- Isolated: Isolated VLAN ports can only send packets to promiscuous ports. Such packets are sent using an isolated VLAN ID.
- 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.
- 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:

- 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.
- 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.
- A packet received on a promiscuous port SHOULD be forwarded out all types of ports in the private VLAN.
- A packet received on an inter-bridge port with an isolated VLAN ID should be forwarded as a packet received on an isolated port.
- 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.
- 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 for 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 avoids 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 promiscious port would work since the host can reach it. However, communication will fail if that port is not promicious. 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 Beebee, 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., Pougnard, 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>.

Author's Address

Erik Nordmark
Arista Networks
Santa Clara, CA
USA

Email: nordmark@arista.com

Internet Area WG
Internet-Draft
Intended status: Informational
Expires: September 14, 2017

C. Perkins
Futurewei
D. Stanley
HPE
W. Kumari
Google
JC. Zuniga
SIGFOX

March 13, 2017

Multicast Considerations over IEEE 802 Wireless Media
draft-perkins-intarea-multicast-ieee802-02

Abstract

Some performance issues have been observed when multicast packet transmissions of IETF protocols are used over IEEE 802 wireless media. Even though enhamcements for multicast transmissions have been designed at both IETF and IEEE 802, there seems to exist a disconnect between specifications, implementations and configuration choices.

This draft describes the different issues that have been observed, the multicast enhancement features that have been specified at IETF and IEEE 802 for wireless media, as well as the operational chioces that can be taken to improve the performace of the network. Finally, it provides some recommendations about the usage and combination of these features and operational choices.

Table of Contents

1. Introduction

Many IETF protocols depend on multicast/broadcast for delivery of control messages to multiple receivers. Multicast is used for various purposes such as neighborhood discovery, network flooding, address resolution, as well as for reduction in media access for the transmission of data that is intended for multiple receivers.

IETF protocols typically rely on network protocol layering in order to reduce or eliminate any dependence of higher level protocols on the specific nature of the MAC layer protocols or the physical media. In the case of multicast transmissions, higher level protocols have traditionally been designed as if transmitting a packet to an IP address had the same cost in interference and network media access, regardless of whether the destination IP address is a unicast address or a multicast or broadcast address. This model was reasonable for networks where the physical medium was usually wired, like Ethernet. Unfortunately, for many wireless media, the costs to access the medium can be quite different. Some enhancements have been designed in IETF protocols that are assumed to work primarily over wilress media. However, these enhancements are usually implemented in limited deployments and not widely spread on most wireless networks.

IEEE 802 wireless protocols have been designed with certain fetures to support multicat traffic. For instance, lower modulations are used to transmit multicast frames, so that these can be received by all stations in the cell, regardless of the distance or path attenuation from the base station or access point. However, these lower modulation transmissions take longer on the medium and therefore they reduce the capabilities to transmit more high efficiency traffic with higher order modulations to stations that may be in closer vicinity. Due to these and other reasons, some IEEE 802 working groups like 802.11 have designed several features to improve the performance of multicast transmissions at Layer 2 [REF 11-15-1261-03]. Besides protocol design features, some operational and configuration enhancements can also be applied to overcome the network performance issues created by multicast traffic.

This Internet Draft identifies the problems created by the usage of multicast traffic over wireless networks. It also highlights the different enhancements that have been designed at IETF and IEEE 802, as well as the operational choices that can be taken, to ameliorate the effects of multicast traffic. Some recommendations about the usage and combinations of these enhancements are also provided.

2. Terminology

This document uses the following definitions:

AP
: IEEE 802.11 Access Point.

STA
: IEEE 802.11 station.

basic rate
: The "lowest common denominator" data rate at which multicast and broadcast traffic is generally transmitted.

MCS
: Modulation and Coding Scheme.

3. Identified mulitcast issues

3.1. Issues at Layer 2 and below

In this section we list some of the issues related to the use of multicast transmissions over IEEE 802 wireless technologies.

3.1.1. Multicast reliability

Multicast traffic is typically much less reliable than unicast traffic. Since multicast makes point-to-multipoint communications, multiple acknowledgements would be needed to guarantee the reception on all recepients.

3.1.2. Lower data rate

Because lower MCS have longer range but also lower data rate, multicast / broadcast traffic is generally transmitted at the lowest common denominator rate, also known as a basic rate. On IEEE 802.11 networks (aka Wi-Fi), this rate might be as low as 6 Mbps, when some unicast links in the same cell can be operating at rates up to 600 Mbps. Transmissions at a lower rate require more occupancy of the wireless medium and thus restrict the airtime for all other medium communications and degrade the overall capacity.

Wired multicast affects wireless LANs because the AP extends the wired segment and multicast / broadcast frames on the wired LAN side are copied to WLAN. Since broadcast messages are transmitted at the most robust MCS, this implies that large frames sent at slow rate over the air.

3.1.3. Power-save effects on multicast

Multicast can work poorly with the power-save mechanisms defined in IEEE 802.11.

- Both unicast and multicast traffic can be delayed by power-saving mechanisms.
- Unicast is delayed until a STA wakes up and asks for it. Additionally, unicast traffic may be delayed to improve power save, efficiency and increase probability of aggregation.
- Multicast traffic is delayed in a wireless network if any of the STAs in that network are power savers. All STAs have to be awake at a known time to receive multicast traffic.
- Packets can also be discarded due to buffer limitations in the AP and non-AP STA.

3.2. Issues at Layer 3 and above

In this section we mention a few representative IETF protocols, and describe some possible negative effects due to performance degradation when using multicast transmissions for control messages. Common uses of multicast include:

- Control plane for IPv4 and IPv6
- ARP and Neighbor Discovery
- Service discovery
- Applications (video delivery, stock data etc)
- Other L3 protocols (non-IP)

3.2.1. IPv4 issues

The following list contains a few representative IPv4 protocols using multicast.

- ARP
- DHCP
- mDNS

After initial configuration, ARP and DHCP occur much less commonly.

3.2.2. IPv6 issues

The following list contains a few representative IPv6 protocols using multicast. IPv6 makes much more extensive use of multicast.

- DHCPv6
- Liveness detection (NUD)

- o Some control plane protocols are not very tolerant of packet loss, especially neighbor discovery.
- o Services may be considered lost if several consecutive packets fail.

Address Resolution

Service Discovery

Route Discovery

Decentralized Address Assignment

Geographic routing

3.2.3. MLD issues

Multicast Listener Discovery(MLD) [RFC4541] is often used to identify members of a multicast group that are connected to the ports of a switch. Forwarding multicast frames into a WiFi-enabled area can use such switch support for hardware forwarding state information. However, since IPv6 makes heavy use of multicast, each STA with an IPv6 address will require state on the switch for several and possibly many multicast solicited-node addresses. Multicast addresses that do not have forwarding state installed (perhaps due to hardware memory limitations on the switch) cause frames to be flooded on all ports of the switch.

3.2.4. Spurious Neighbor Discovery

On the Internet there is a "background radiation" of scanning traffic (people scanning for vulnerable machines) and backscatter (responses from spoofed traffic, etc). This means that the router is constantly getting packets destined for machines whose IP addresses may or may not be in use. In the cases where the IP is assigned to a machine, the router broadcasts an ARP request, gets back an ARP reply, caches this and then can deliver traffic to the host. In the cases where the IP address is not in use, the router broadcasts one (or more) ARP requests, and never gets a reply. This means that it does not populate the ARP cache, and the next time there is traffic for that IP address it will broadcast ARP requests again.

The rate of these ARP requests is proportional to the size of the subnets, the rate of scanning and backscatter, and how long the router keeps state on non-responding ARPs. As it turns out, this rate is inversely proportional to how occupied the subnet is (valid ARPs end up in a cache, stopping the broadcasting; unused IPs never respond, and so cause more broadcasts). Depending on the address

space in use, the time of day, how occupied the subnet is, and other unknown factors, on the order of 2000 broadcasts per second have been observed at the IETF NOCs.

On a wired network, there is not a huge difference amongst unicast, multicast and broadcast traffic; but this is not true in the wireless realm. Wireless equipment often is unable to send this amount of broadcast and multicast traffic. Consequently, on the wireless networks, we observe a significant amount of dropped broadcast and multicast packets. This, in turn, means that when a host connects it is often not able to complete DHCP, and IPv6 RAs get dropped, leading to users being unable to use the network.

4. Multicast protocol optimizations

This section lists some optimizations that have been specified in IEEE 802 and IETF that are aimed at reducing or eliminating the issues discussed in Section 3.

4.1. Proxy ARP in 802.11-2012

The AP knows all associated STAs MAC address and IP address; in other words, the AP acts as the central "manager" for all the 802.11 STAs in its BSS. Proxy ARP is easy to implement at the AP, and offers the following advantages:

- Reduced broadcast traffic (transmitted at low MCS) on the wireless medium
- STA benefits from extended power save in sleep mode, as ARP requests are replied to by AP.
- Keeps ARP frames off the wireless medium.
- Changes are not needed to STA implementation.

Here is the specification language from clause 10.23.13 in [2] as described in [dot11-proxyarp]:

> When the AP supports Proxy ARP "[...] the AP shall maintain a Hardware Address to Internet Address mapping for each associated station, and shall update the mapping when the Internet Address of the associated station changes. When the IPv4 address being resolved in the ARP request packet is used by a non-AP STA currently associated to the BSS, the proxy ARP service shall respond on behalf of the non-AP STA"

4.2. Buffering to improve Power-Save

The AP acts on behalf of STAs in various ways. In order to improve the power-saving feature for STAs in its BSS, the AP buffers frames for delivery to the STA at the time when the STA is scheduled for reception.

4.3. IPv6 support in 802.11-2012

IPv6 uses Neighbor Discovery Protocol (NDP) instead Every IPv6 node subscribes to special multicast address Neighbor-Solicitation message replaces ARP

Here is the specification language from-10.23.13 in [2]:

> "When an IPv6 address is being resolved, the Proxy Neighbor Discovery service shall respond with a Neighbor Advertisement message [...] on behalf of an associated STA to an [ICMPv6] Neighbor Solicitation message [...]. When MAC address mappings change, the AP may send unsolicited Neighbor Advertisement Messages on behalf of a STA."

NDP may be used to request additional information

- Maximum Transmission Unit
- Router Solicitation
- Router Advertisement, etc.

NDP messages are sent as group addressed (broadcast) frames in 802.11. Using the proxy operation helps to keep NDP messages off the wireless medium.

4.4. Conversion of multicast to unicast

It is often possible to transmit control and data messages by using unicast transmissions to each station individually.

4.5. Directed Multicast Service (DMS)

There are situations where more is needed than simply converting multicast to unicast [Editor’s note: citation needed]. For these purposes, DMS enables a client to request that the AP transmit multicast group addressed frames destined to the requesting clients as individually addressed frames [i.e., convert multicast to unicast].

- DMS Requires 802.11n A-MSDUs

- Individually addressed frames are acknowledged and are buffered for power save clients
- Requesting STA may specify traffic characteristics for DMS traffic
- DMS was defined in IEEE Std 802.11v-2011

DMS is not currently implemented in products. DMS does require changes to both AP and STA implementation.

4.6. GroupCast with Retries (GCR)

GCR (defined in [dot11aa]) provides greater reliability by using either unsolicited retries or a block acknowledgement mechanism. GCR increases probability of broadcast frame reception success, but still does not guarantee success.

For the block acknowledgement mechanism, the AP transmits each group addressed frame as conventional group addressed transmission. Retransmissions are group addressed, but hidden from non-11aa clients. A directed block acknowledgement scheme is used to harvest reception status from receivers; retransmissions are based upon these responses.

GCR is suitable for all group sizes including medium to large groups. As the number of devices in the group increases, GCR can send block acknowledgement requests to only a small subset of the group. GCR does require changes to both AP and STA implementation.

GCR may introduce unacceptable latency. After sending a group of data frames to the group, the AP has do the following:

- unicast a Block Ack Request (BAR) to a subset of members.
- wait for the corresponding Block Ack (BA).
- retransmit any missed frames.
- resume other operations which may have been delayed.

This latency may not be acceptable for some traffic.

There are ongoing extensions in 802.11 to improve GCR performance.

- BAR is sent using downlink MU-MIMO (note that downlink MU-MIMO is already specified in 802.11-REVmc 4.3).
- BA is sent using uplink MU-MIMO (which is a .11ax feature).
- Additional 802.11ax extensions are under consideration; see [mc-ack-mux]
- Latency may also be reduced by simultaneously receiving BA information from multiple clients.

5. Operational optimizations

This section lists some operational optimizations that can be implemented when deploying wireless IEEE 802 networks to mitigate the issues discussed in Section 3.

5.1. Mitigating Problems from Spurious Neighbor Discovery

ARP Sponges

An ARP Sponge sits on a network and learn which IPs addresses are actually in use. It also listen for ARP requests, and, if it sees an ARP for an IP address which it believes is not used, it will reply with its own MAC address. This means that the router now has an IP to MAC mapping, which it caches. If that IP is later assigned to an machine (e.g using DHCP), the ARP sponge will see this, and will stop replying for that address. Gratuitous ARPs (or the machine ARPing for its gateway) will replace the sponged address in the router ARP table. This technique is quite effective; but, unfortunately, the ARP sponge daemons were not really designed for this use (the standard one [arpsponge], was designed to deal with the disappearance of participants from an IXP) and so are not optimized for this purpose. We have to run one daemon per subnet, the tuning is tricky (the scanning rate versus the population rate versus retires, etc.) and sometimes the daemons just seem to stop, requiring a restart of the daemon and causing disruption.

Router mitigations

Some routers (often those based on Linux) implement a "negative ARP cache" daemon. Simply put, if the router does not see a reply to an ARP it can be configured to cache this information for some interval. Unfortunately, the core routers which we are using do not support this. When a host connects to network and gets an IP address, it will ARP for its default gateway (the router). The router will update its cache with the IP to host MAC mapping learnt from the request (passive ARP learning).

Firewall unused space

The distribution of users on wireless networks / subnets changes from meeting to meeting (e.g the "IETF-secure" SSID was renamed to "IETF", fewer users use "IETF-legacy", etc). This utilization is difficult to predict ahead of time, but we can monitor the usage as attendees use the different networks. By

configuring multiple DHCP pools per subnet, and enabling them sequentially, we can have a large subnet, but only assign addresses from the lower portions of it. This means that we can apply input IP access lists, which deny traffic to the upper, unused portions. This means that the router does not attempt to forward packets to the unused portions of the subnets, and so does not ARP for it. This method has proven to be very effective, but is somewhat of a blunt axe, is fairly labor intensive, and requires coordination.

Disabling/filtering ARP requests

In general, the router does not need to ARP for hosts; when a host connects, the router can learn the IP to MAC mapping from the ARP request sent by that host. This means that we should be able to disable and / or filter ARP requests from the router. Unfortunately, ARP is a very low level / fundamental part of the IP stack, and is often offloaded from the normal control plane. While many routers can filter layer-2 traffic, this is usually implemented as an input filter and / or has limited ability to filter output broadcast traffic. This means that the simple "just disable ARP or filter it outbound" seems like a really simple (and obvious) solution, but implementations / architectural issues make this difficult or awkward in practice.

NAT

The broadcasts are overwhelmingly being caused by outside scanning / backscatter traffic. This means that, if we were to NAT the entire (or a large portion) of the attendee networks, there would be no NAT translation entries for unused addresses, and so the router would never ARP for them. The IETF NOC has discussed NATing the entire (or large portions) attendee address space, but a: elegance and b: flaming torches and pitchfork concerns means we have not attempted this yet.

Stateful firewalls

Another obvious solution would be to put a stateful firewall between the wireless network and the Internet. This firewall would block incoming traffic not associated with an outbound request. The IETF philosophy has been to have the network as open as possible / honor the end-to-end principle. An attendee on the meeting network should be an Internet host, and should be able to receive unsolicited requests. Unfortunately, keeping the network working and stable is the first priority

and a stateful firewall may be required in order to achieve this.

6. Multicast Considerations for Other Wireless Media

Many of the causes of performance degradation described in earlier sections are also observable for wireless media other than 802.11.

For instance, problems with power save, excess media occupancy, and poor reliability will also affect 802.15.3 and 802.15.4. However, 802.15 media specifications do not include similar mechanisms of the type that have been developed for 802.11. In fact, the design philosophy for 802.15 is more oriented towards minimality, with the result that many such functions would more likely be relegated to operation within higher layer protocols. This leads to a patchwork of non-interoperable and vendor-specific solutions. See [uli] for some additional discussion, and a proposal for a task group to resolve similar issues, in which the multicast problems might be considered for mitigation.

7. Recommendations

This section provides some recommendations about the usage and combinations of the multicast enhancements described in Section 4 and Section 5.

(FFS)

8. Security Considerations

This document does not introduce any security mechanisms, and does not have any impact on existing security mechanisms.

9. IANA Considerations

This document does not specify any IANA actions.

10. Informative References

[arpsponge] Arien Vijn, Steven Bakker, , "Arp Sponge", March 2015.

[dot11] P802.11, , "Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications", March 2012.

[dot11-proxyarp] P802.11, , "Proxy ARP in 802.11ax", September 2015.

[dot11aa] P802.11, , "Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 2: MAC Enhancements for Robust Audio Video Streaming", March 2012.

[mc-ack-mux] Yusuke Tanaka et al., , "Multiplexing of Acknowledgements for Multicast Transmission", July 2015.

[mc-prob-stmt] Mikael Abrahamsson and Adrian Stephens, , "Multicast on 802.11", March 2015.

[mc-props] Adrian Stephens, , "IEEE 802.11 multicast properties", March 2015.

[RFC4541] Christensen, M., Kimball, K., and F. Solensky, "Considerations for Internet Group Management Protocol (IGMP) and Multicast Listener Discovery (MLD) Snooping Switches", RFC 4541, DOI 10.17487/RFC4541, May 2006, <http://www.rfc-editor.org/info/rfc4541>.

[uli] Pat Kinney, , "LLC Proposal for 802.15.4", Nov 2015.

Authors' Addresses

Charles E. Perkins
Futurewei Inc.
2330 Central Expressway
Santa Clara, CA 95050
USA

Phone: +1-408-330-4586
Email: charliep@computer.org

Dorothy Stanley
Hewlett Packard Enterprise
2000 North Naperville Rd.
Naperville, IL 60566
USA

Phone: +1 630 979 1572
Email: dstanley@arubanetworks.com

Warren Kumari
Google
1600 Amphitheatre Parkway
Mountain View, CA 94043
USA

Email: warren@kumari.net

Juan Carlos Zuniga
SIGFOX
425 rue Jean Rostand
Labege 31670
France

Email: j.c.zuniga@ieee.org

INTAREA
Internet-Draft
Intended status: Standards Track
Expires: September 14, 2017

S. Kanugovi
S. Vasudevan
Nokia
J. Zhu
Intel
F. Baboescu
Broadcom
S. Peng
Huawei

March 13, 2017

Control Plane Protocols and Procedures for Multiple Access Management Services
draft-zhu-intarea-mams-control-protocol-00

Abstract

Today, a device can be simultaneously connected to multiple communication networks based on different technology implementations and network architectures like WiFi, LTE, DSL. In such multi-connectivity scenario, it is desirable to combine multiple access networks or select the best one to improve quality of experience for a user and improve overall network utilization and efficiency. This document presents the control plane protocols, as well as describes control plane procedures for configuring the user plane in a multi access management services (MAMS) framework that can be used to flexibly select the combination of uplink and downlink access and core network paths, and user plane treatment for improving network efficiency and enhanced application quality of experience.

Table of Contents

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 Service (MAMS) [I-D.kanugovi-intarea-mams-protocol] is a framework to select and configure network paths when multiple connections can serve a client device. It allows the path selection and configuration to adapt to dynamic network conditions. It is based on principles of user plane interworking that enables the solution to be deployed as an overlay without impacting the underlying networks.

This document presents the control plane protocols for the MAMS framework. It co-exists and complements user plane protocols (e.g. MPTCP [RFC6824] or MPTCP Proxy [I-D.boucadair-mptcp-plain-mode], [I-D.wei-mptcp-proxy-mechanism]) by providing a way to negotiate and configure them based on client and network capabilities. It allows exchange of network state information and leverages network intelligence to optimize the performance of such protocols.

3. Terminology

"Anchor Connection": Refers to the network path from the N-MADP to the Application Server that corresponds to a specific IP anchor that has assigned an IP address to the client.

"Delivery Connection": Refers to the network path from the N-MADP to the client.

"Network Connection Manager" (NCM), "Client Connection Manager" (CCM), "Network Multi Access Data Proxy" (N-MADP), and "Client Multi Access Data Proxy" (C-MADP) in this document are to be interpreted as described in [I-D.kanugovi-intarea-mams-protocol].

4. MAMS Control-Plane Protocol

4.1. Overview

The MAMS architecture [I-D.kanugovi-intarea-mams-protocol] introduces the following functional elements,

- Network Connection Manager (NCM) and Client Connection Manager (CCM) in the control plane, and

o Network Multi Access Data Proxy (N-MADP) and Client Multi Access Data Proxy (C-MADP) handling the user plane.

Figure 1 shows the default MAMS control plane protocol stack. HTTPS is used for transporting management and control messages between NCM and CCM.

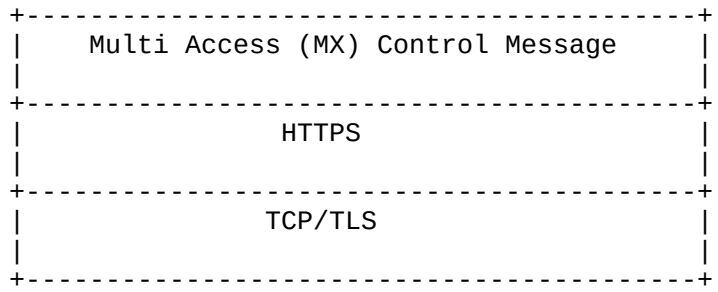


Figure 1: TCP-based MAMS Control Plane Protocol Stack

5. MAMS User Plane Protocol

Figure 2 shows the MAMS user plane protocol stack.

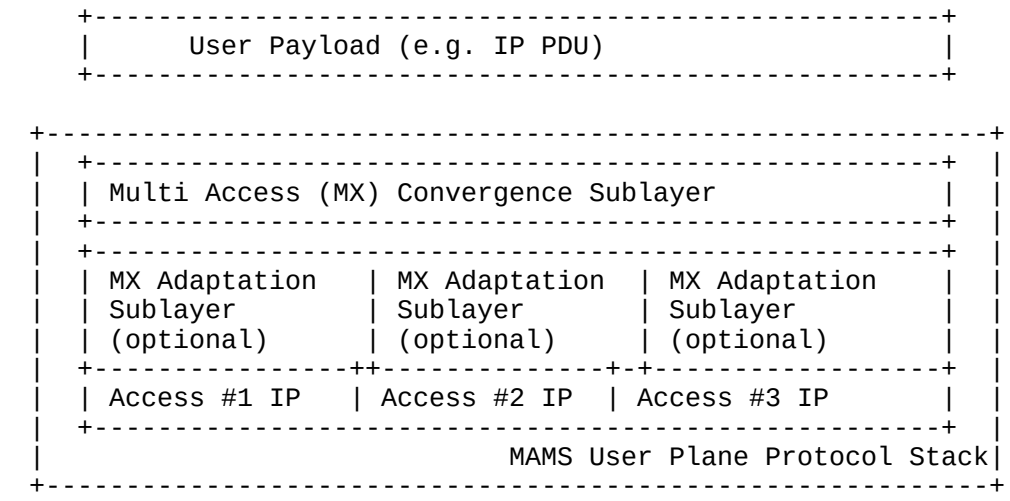


Figure 2: MAMS User Plane Protocol Stack

It consists of the following two Sublayers:

- Multi-Access (MX) Convergence Sublayer: This layer performs multi-access specific tasks, e.g. access (path) selection, multi-link (path) aggregation, splitting/reordering, lossless switching, fragmentation, concatenation, etc. For example, MX Convergence layer can be implemented using existing user plane protocols like MPTCP or by adapting encapsulating header/trailer schemes (e.g Trailer Based MX Convergence as specified in [I-D.zhu-intarea-mams-user-protocol]).
- Multi-Access (MX) Adaptation Sublayer: This layer performs functions to handle tunnelling, network layer security, and NAT. For example, MX Adaptation can be implemented using IPsec, DTLS or Client NAT (Source NAT at Client with inverse mapping at N-MADP [I-D.zhu-intarea-mams-user-protocol]). The MX Adaptation Layer is optional and can be independently configured for each of the Access Links, e.g. in a deployment with LTE (assumed secure) and Wi-Fi (assumed not secure), the MX Adaptation Sublayer can be omitted for the LTE link but MX Adaptation Sublayer is configured as IPsec for the Wi-Fi link.

6. MAMS Control Plane Procedures

6.1. Overview

CCM and NCM exchange signaling messages to configure the user plane functions, C-MADP and N-MADP, at the client and network respectively. The means for CCM to obtain the NCM credentials (FQDN or IP Address) for sending the initial discovery messages are outside of the scope of MAMS document, e.g. using methods like provisioning, DNS. Once the discovery process is successful, the (initial) NCM can update and assign additional NCM addresses for sending subsequent control plane messages.

CCM discovers and exchanges capabilities with the NCM. NCM provides the credentials of the N-MADP end-point and negotiates the parameters for user plane with the CCM. CCM configures C-MADP to setup the user plane path (e.g. MPTCP/UDP Proxy Connection) with the N-MADP based on the credentials (e.g. (MPTCP/UDP) Proxy IP address and port, Associated Core Network Path), and the parameters exchanged with the NCM. The key procedures are described in details in the following sub-sections.

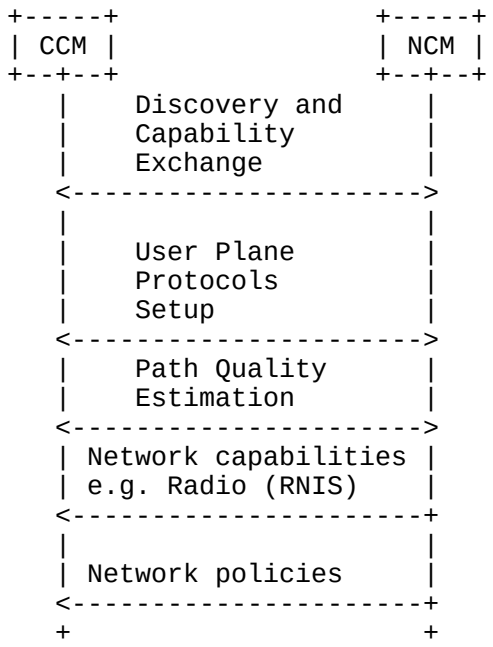


Figure 3: MAMS Control Plane Procedures

6.2. Common fields in MAMS Control Messages

Each MAMS control message consists of the following common fields:

- o Version: indicates the version of MAMS control protocol.
- o Message Type: indicates the type of the message, e.g. MX Discovery, MX Capability REQ/RSP etc.
- o Sequence Number: auto-incremented integer to uniquely identify a transaction of message exchange, e.g. MX Capability REQ/RSP.

6.3. Discovery & Capability Exchange

Figure 4 shows the MAMS discovery and capability exchange procedure consisting of the following key steps:

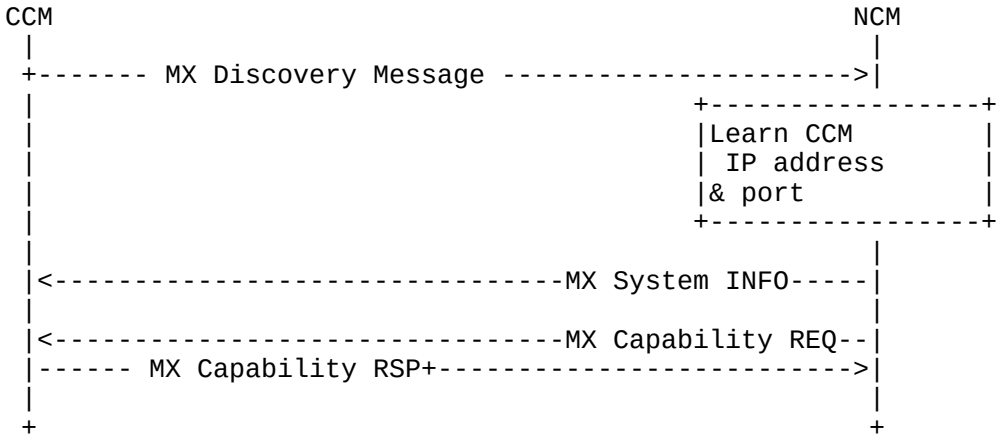


Figure 4: MAMS Control Procedure for Discovery & Capability Exchange

Step 1 (Discovery): CCM periodically sends out the MX Discovery Message to a pre-defined (NCM) IP Address/ port until receives an MX System INFO message in acknowledgement.

MX Discovery Message includes the following information:

o MAMS Version

MX System INFO includes the following information:

o Number of Anchor Connections

  For each Anchor Connection, it includes the following parameters:

  * Connection ID: Unique identifier for the Anchor Connection
  * Connection Type (e.g., 0: Wi-Fi; 1: 5G NR; 2: Multi-Fire; 3: LTE)
  * NCM Endpoint Address (For Control Plane Messages over this connection)

    + IP Address or FQDN (Fully Qualified Domain Name)
    + Port Number

Step 2 (Capability Exchange): once receiving a MX discovery message, NCM learns the IP address and port number to communicate with CCM, and sends out the MX Capability REQ message, including the following Parameters:

- MX Feature Activation List: Indicates if the corresponding feature is enabled or not, e.g. lossless switching, fragmentation, concatenation, Uplink aggregation, Downlink aggregation, Measurement, etc.
- Number of Anchor Connections (Core Networks)

  For each Anchor Connection, it includes the following parameters:

  * Connection ID
  * Connection Type (e.g., 0: Wi-Fi; 1: 5G NR; 2: Multi-Fire; 3: LTE)
- Number of Delivery Connections (Access Links)

  For each Delivery Connection, it includes the following parameters:

  * Connection ID
  * Connection Type (e.g., 0: Wi-Fi; 1: 5G NR; 2: Multi-Fire; 3: LTE)
- MX Convergence Method Support List

  * Trailer-based MX Convergence;
  * MPTCP Proxy;
- MX Adaptation Method Support List

  * UDP Tunnel without DTLS;
  * UDP Tunnel with DTLS;
  * IPsec Tunnel[RFC3948];
  * Client NAT;

In response, CCM sends out the MX Capability RSP message, including the following information:

- MX Feature Activation List: Indicates if the corresponding feature is enabled or not, e.g. lossless switching, fragmentation, concatenation, Uplink aggregation, Downlink aggregation, Measurement, etc.
- Number of Anchor Connections (Core Networks)

  For each Anchor Connection, it includes the following parameters:

  * Connection ID
  * Connection Type (e.g., 0: Wi-Fi; 1: 5G NR; 2: Multi-Fire; 3: LTE)
- Number of Delivery Connections (Access Links)

  For each Delivery Connection, it includes the following parameters:

      * Connection ID
      * Connection Type (e.g., 0: Wi-Fi; 1: 5G NR; 2: Multi-Fire; 3: LTE)
   o MX Convergence Method Support List

      * Trailer-based MX Convergence;
      * MPTCP Proxy;
   o MX Adaptation Method Support List

      * UDP Tunnel without DTLS;
      * UDP Tunnel with DTLS;
      * IPsec Tunnel[RFC3948];
      * Client NAT;

6.4. User Plane Configuration

Figure 5 shows the user plane configuration procedure consisting of the following key steps:

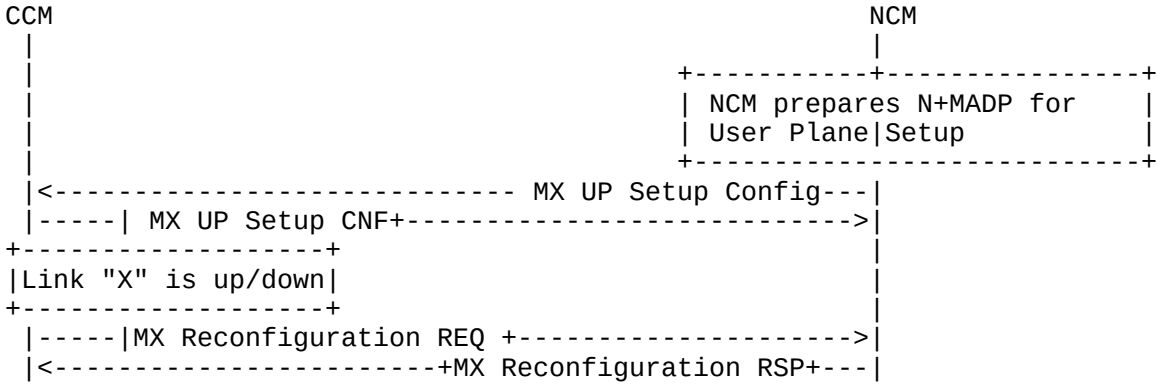


Figure 5: MAMS Control Procedure for User Plane Configuration

User Plane Protocols Setup: Based on the negotiated capabilities, NCM sets up the user plane (Adaptation Layer and Convergence Layer) protocols at the N-MADP, and informs the CCM of the user plane protocols to setup at the client (C-MADP) and the parameters for C-MADP to connect to N-MADP.

Each MADP instance is responsible for one anchor connection. The MX UP Setup Config consists of the following parameters:

- o Number of Anchor Connections (Core Networks)

  For Each Anchor Connection, it includes the following parameters

  - * Anchor Connection ID
  - * Connection Type (e.g., 0: Wi-Fi; 1: 5G NGC; 2: Multi-Fire; 3: LTE)
  - * MX Convergence Method
    - + Trailer-based MX Convergence;
    - + MPTCP Proxy;
  - * MX Convergence Method Parameters
    - + Convergence Proxy IP Address
    - + Convergence Proxy Port
  - * Number of Delivery Connections

    For each Delivery Connection, include the following:

    - + Delivery Connection ID
    - + Connection Type (e.g., 0: Wi-Fi; 1: 5G NGC; 2: Multi-Fire; 3: LTE)
    - + MX Adaptation Method
      - - UDP Tunnel without DTLS;
      - - UDP Tunnel with DTLS;
      - - IPSec Tunnel;
      - - Client NAT;
    - + MX Adaptation Method Parameters
      - - Tunnel Endpoint IP Address
      - - Tunnel Endpoint Port
      - - Shared Secret

e.g. When LTE and Wi-Fi are the two user plane accesses, NCM conveys to CCM that IPsec needs to be setup as the MX Adaptation Layer over the Wi-Fi Access, using the following parameters - IPsec end-point IP address, Pre-Shared Key., No Adaptation Layer is needed over the LTE Access as it is considered secure with no NAT. The MX Convergence Method is configured as MPTCP Proxy along with parameters for connection to the MPTCP Proxy, namely IP Address and Port of the MPTCP Proxy for TCP Applications.

Once the user plane protocols are configured, CCM informs the NCM of the status via the MX UP Setup CNF message

Reconfiguration: when the client detects that the link is up/down or the IP address changes (e.g. via APIs provided by the client OS), CCM

sends out a MX Reconfiguration REQ Message to setup / release /update the connection, and the message SHOULD include the following information

- Reconfiguration Action: indicate the reconfiguration action (0: release; 1: setup; 2: update)
- Connection ID: identify the connection for reconfiguration

If (Reconfiguration Action is setup or update), then include the following parameters

- IP address of the connection
- MTU (Maximum Transmission Unit) size of the connection

6.5. MAMS Path Quality Estimation

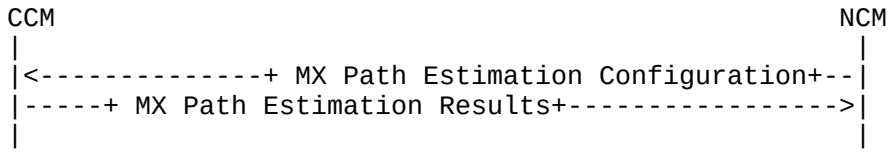


Figure 6: MAMS Control Plane Procedure for Path Quality Estimation

NCM sends following the configuration parameters in the MX Path Estimation Configuration message to the CCM

- Connection ID (of Delivery Connection whose path quality needs to be estimated)
- Init Probe Test Duration (ms)
- Init Probe Test Rate (Mbps)
- Init Probe Size (Bytes)
- Init Probe Ack Required (0 -> No/1 -> Yes)
- Active Probe Frequency (ms)
- Active Probe Size (Bytes)
- Active Probe Ack Required (0 -> No/1 -> Yes)

CCM configures the C-MADP for probe reception based on these parameters and for collection of the statistics according to the following configuration.

- Init Probe Results Configuration

    * Lost Probes (%)
    * Probe Delay
    * Probe Rate
  o Active Probe Results Configuration

    * Average Throughput in the last Probe Duration

The user plane probing is divided into two phases - Initialization phase and Active phase.

- o Initialization phase: A network path that is not included by N-MADP for transmission of user data is deemed to be in the Initialization phase. The user data may be transmitted over other available network paths.
- o Active phase: A network path that is included by N-MADP for transmission of user data is deemed to be in Active phase.

In Initialization phase, NCM configures N-MADP to send an MX Idle Probe REQ message. CCM collects the Idle probe statistics from C-MADP and sends the MX Path Estimation Results Message to NCM per the Initialization Probe Results configuration.

In Active phase, NCM configures N-MADP to send an MX Active Probe REQ message.. C-MADP calculates the metrics as specified by the Active Probe Results Configuration. CCM collects the Active probe statistics from C-MADP and sends the MX Path Estimation Results Message to NCM per the Active Probe Results configuration.

6.6. MAMS Traffic Steering

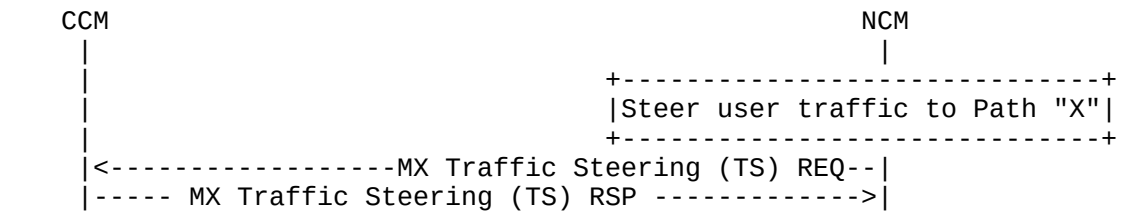


Figure 7: MAMS Traffic Steering Procedure

NCM sends out a MX Traffic Steering (TS) REQ message to steer data traffic. It is also possible to send data traffic over multiple connections simultaneously, i.e. aggregation. The message includes the following information:

- o Connection ID of the Anchor Connection

- o Connection ID List of Delivery Connections for DL traffic
- o Connection ID List of Delivery connections for UL traffic
- o MX Feature Activation List: each parameter indicates if the corresponding feature is enabled or not: lossless switching, fragmentation, concatenation, Uplink aggregation, Downlink aggregation, Measurement

In response, CCM sends out a MX Traffic Steering (TS) RSP message, including the following information:

- o MX Feature Activation List: each parameter indicates if the corresponding feature is enabled or not: lossless switching, fragmentation, concatenation, Uplink aggregation, Downlink aggregation

7. Applying MAMS Control Procedures with MPTCP Proxy as User Plane

If NCM determines that N-MADP is to be instantiated with MPTCP as the MX Convergence Protocol, it exchanges the MPTCP capability support in discovery and capability exchange procedures. NCM then exchanges the credentials of the N-MADP instance, setup as MPTCP Proxy, along with related parameters to the CCM. CCM configures C-MADP with these parameters to connect with the N-MADP (MPTCP proxy [I-D.wei-mptcp-proxy-mechanism], [I-D.boucadair-mptcp-plain-mode]) instance, on the available network path (Access).

Figure 8 shows the MAMS assisted MPTCP Proxy control procedure.

- o For securing the TCP subflow data over links that cannot be assumed to be secure, NCM configures MX Adaptation Layer. E.g. NCM can inform CCM to use IPsec as the MX Adaptation Layer over the link "X" (e.g. Wi-Fi). CCM informs C-MADP to set up IPSec (transport mode) with N-MADP using the MPTCP-Proxy IP address to protect the TCP subflow over Link "X".
- o NCM informs the CCM that N-MADP is configured as the MPTCP proxy and provides the parameters like MPTCP Proxy IP address/Port. C-MADP obtains the IP address & port of MPTCP-Proxy for Link "X" locally from CCM. This is useful if N-MADP is reachable via different IP address or/and port, from different access networks. The current MPTCP signaling can’t identify or differentiate the MPTCP proxy IP address & port among multiple access networks.

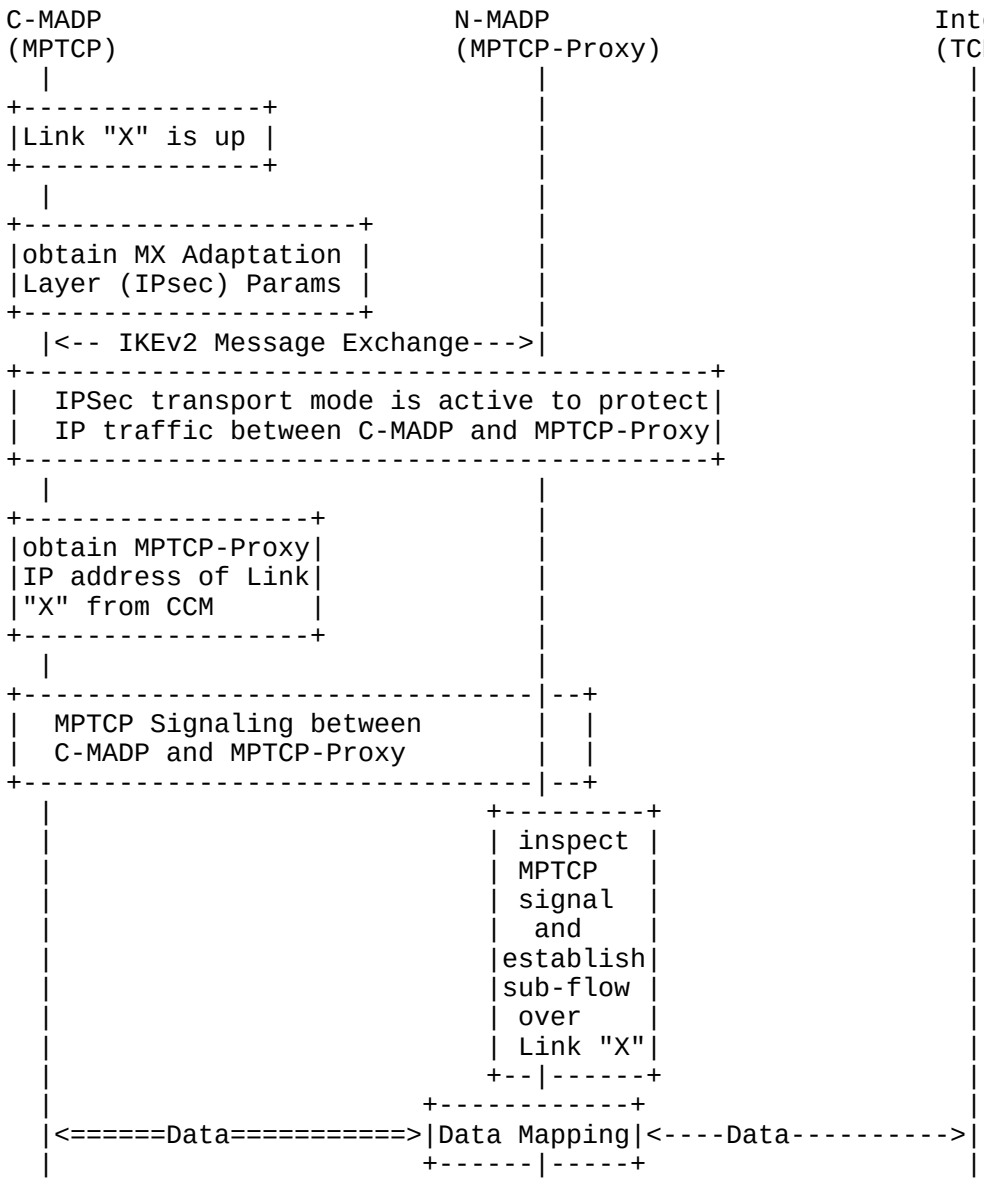


Figure 8: MAMS-assisted MPTCP Proxy as User Plane

8. Co-existence of MX Adaptation and MX Convergence Layers

MAMS u-plane protocols support multiple combinations and instances of user plane protocols to be used in the MX Adaptation and the Convergence layer.

For example, one instance of the MX Convergence Layer can be MPTCP Proxy and another instance can be Trailer based. The MX Adaptation for each can be either UDP tunnel or IPsec. IPSec may be set up when network pathneeds to be secured, e.g. to protect the TCP subflow traversing the network path between the client and MPTCP proxy.

Each of the instances of MAMS user plane, i.e. combination of MX Convergence and MX Adaptation layer protocols, can coexist simultaneously and independently handle different traffic types.

9. Security Considerations

9.1. MAMS Control plane security

For deployment scenarios, where the client is configured (e.g. by the network operator) to use a specific network for exchanging control plane messages and assume the network path to be secure, MAMS control messages will rely on security provided by the underlying transport network.

For deployment scenarios where the security of the network path cannot be assumed, NCM and CCM implementations MUST support the "https" URI scheme [RFC2818] and Transport Layer Security (TLS) [RFC5246] to secure control plane message exchange between the NCM and CCM.

For deployment scenarios where client authentication is desired, HTTP Digest Authentication MUST be supported. TLS Client Authentication is the preferred mechanism if it is available.

9.2. MAMS User plane security

User data in MAMS framework relies on the security of the underlying network transport paths. When this cannot be assumed, NCM configures use of protocols, like IPsec [RFC4301] [RFC3948] in the MX Adaptation Layer, for security.

10. Contributing Authors

The editors gratefully acknowledge the following additional contributors in alphabetical order: A Krishna Pramod/Nokia, Hannu Flinck/Nokia, Hema Pentakota/Nokia, Nurit Sprecher/Nokia

11. References

11.1. 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>.

11.2. Informative References

[I-D.boucadair-mptcp-plain-mode] Boucadair, M., Jacquenet, C., Bonaventure, O., Behaghel, D., stefano.secci@lip6.fr, s., Henderickx, W., Skog, R., Vinapamula, S., Seo, S., Cloetens, W., Meyer, U., Contreras, L., and B. Peirens, "Extensions for Network-Assisted MPTCP Deployment Models", draft-boucadair-mptcp-plain-mode-10 (work in progress), March 2017.

[I-D.kanugovi-intarea-mams-protocol] Kanugovi, S., Vasudevan, S., Baboescu, F., Zhu, J., Peng, S., and J. Mueller, "Multiple Access Management Services", draft-kanugovi-intarea-mams-protocol-03 (work in progress), March 2017.

[I-D.wei-mptcp-proxy-mechanism] Wei, X., Xiong, C., and E. Ed, "MPTCP proxy mechanisms", draft-wei-mptcp-proxy-mechanism-02 (work in progress), June 2015.

[I-D.zhu-intarea-mams-user-protocol] Zhu, J., "User-Plane Protocols for Multiple Access Management Service", draft-zhu-intarea-mams-user-protocol-00 (work in progress), March 2017.

[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>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

[RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <http://www.rfc-editor.org/info/rfc6824>.

[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>.

Appendix A. MAMS Control Plane Optimization over Secure Connections

If the connection between CCM and NCM over which the MAMS control plane messages are transported is assumed to be secure, UDP is used as the transport for management & control messages between NCM and UCM (see Figure 9).

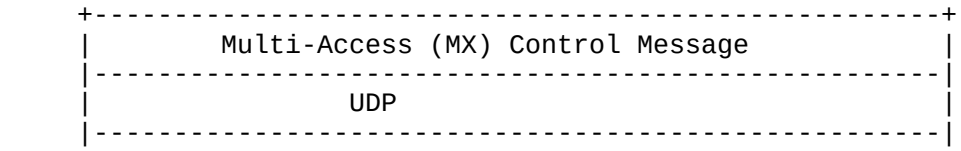


Figure 9: UDP-based MAMS Control plane Protocol Stack

Authors' Addresses

Satish Kanugovi
Nokia

Email: satish.k@nokia.com

Subramanian Vasudevan
Nokia

Email: vasu.vasudevan@nokia.com

Jing Zhu
Intel

Email: jing.z.zhu@intel.com

Florin Baboescu
Broadcom

Email: florin.baboescu@broadcom.com

Shuping Peng
Huawei

Email: pengshuping@huawei.com