

Internet Area WG
Internet-Draft
Intended status: Best Current Practice
Expires: September 5, 2018

R. Bonica
Juniper Networks
F. Baker
Unaffiliated
G. Huston
APNIC
R. Hinden
Check Point Software
O. Troan
Cisco
F. Gont
SI6 Networks
March 4, 2018

IP Fragmentation Considered Fragile
draft-bonica-intarea-frag-fragile-01

Abstract

This document provides an overview of IP fragmentation. It explains how IP fragmentation works and why it is required. As part of that explanation, this document also explains how IP fragmentation reduces the reliability of Internet communication.

This document also proposes alternatives to IP fragmentation. Finally, it provides recommendations for application developers and network operators.

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 <https://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 5, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
2. IP Fragmentation	3
2.1. Links, Paths, MTU and PMTU	3
2.2. Upper-layer Protocols	5
3. Requirements Language	7
4. IP Fragmentation Reduces Reliability	7
4.1. Middle Box Failures	7
4.2. Partial Filtering	8
4.3. Suboptimal Load Balancing	8
4.4. Security Vulnerabilities	9
4.5. Blackholing Due to ICMP Loss	11
4.6. Blackholing Due To Filtering	12
5. Alternatives to IP Fragmentation	12
5.1. Transport Layer Solutions	13
5.2. Application Layer Solutions	14
6. Applications That Rely on IPv6 Fragmentation	15
6.1. DNS	15
6.2. OSPFv3	15
6.3. IP Encapsulations	16
7. Recommendation	16
7.1. For Application Developers	16
7.2. For Network Operators	16
8. IANA Considerations	16
9. Security Considerations	16
10. Acknowledgements	16
11. References	17
11.1. Normative References	17
11.2. Informative References	18
Appendix A. Contributors' Address	20
Authors' Addresses	20

1. Introduction

Operational experience [RFC7872] [Huston] reveals that IP fragmentation reduces the reliability of Internet communication. This document provides an overview of IP fragmentation. It explains how IP fragmentation works and why it is required. As part of that explanation, this document also explains how IP fragmentation reduces the reliability of Internet communication.

This document also proposes alternatives to IP fragmentation. Finally, it provides recommendations for application developers and network operators.

2. IP Fragmentation

2.1. Links, Paths, MTU and PMTU

An Internet path connects a source node to a destination node. A path can contain links and intermediate systems. If a path contains more than one link, the links are connected in series and an intermediate system connects each link to the next. An intermediate system can be a router or a middle box.

Internet paths are dynamic. Assume that the path from one node to another contains a set of links and intermediate systems. If the network topology changes, that path can also change so that it includes a different set of links and intermediate systems.

Each link is constrained by the number of bytes that it can convey in a single IP packet. This constraint is called the link Maximum Transmission Unit (MTU). IPv4 [RFC0791] requires every link to have an MTU of 68 bytes or greater. IPv6 [RFC8200] requires every link to have an MTU of 1280 bytes or greater. These are called the IPv4 and IPv6 minimum link MTU's.

Each Internet path is constrained by the number of bytes that it can convey in a IP single packet. This constraint is called the Path MTU (PMTU). For any given path, the PMTU is equal to the smallest of its link MTU's. Because Internet paths are dynamic, PMTU is also dynamic.

For reasons described below, source nodes estimate the PMTU between themselves and destination nodes. A source node can produce extremely conservative PMTU estimates in which:

- o The estimate for each IPv4 path is equal to IPv4 minimum link MTU (68 bytes).

- o The estimate for each IPv6 path is equal to the IPv6 minimum link MTU (1280 bytes).

While these conservative estimates are guaranteed to be less than or equal to the actual MTU, they are likely to be much less than the actual PMTU. This may adversely affect upper-layer protocol performance.

By executing Path MTU Discovery (PMTUD) [RFC1191] [RFC8201] procedures, a source node can maintain a less conservative, running estimate of the PMTU between itself and a destination node. According to these procedures, the source node produces an initial PMTU estimate. This initial estimate is equal to the MTU of the first link along the path to the destination node. It can be greater than the actual PMTU.

Having produced an initial PMTU estimate, the source node sends non-fragmentable IP packets to the destination node. If one of these packets is larger than the actual PMTU, a downstream router will not be able to forward the packet through the next link along the path. Therefore, the downstream router drops the packet and send an Internet Control Message Protocol (ICMP) [RFC0792] [RFC4443] Packet Too Big (PTB) message to the source node. The ICMP PTB message indicates the MTU of the link through which the packet could not be forwarded. The source node uses this information to refine its PMTU estimate.

PMTUD produces a running estimate of the PMTU between a source node and a destination node. Because PMTU is dynamic, at any given time, the PMTU estimate can differ from the actual PMTU. In order to detect PMTU increases, PMTUD occasionally resets the PMTU estimate to the MTU of the first link along path to the destination node. It then repeats the procedure described above.

Furthermore, PMTUD has the following characteristics:

- o It relies on the network's ability to deliver ICMP PTB messages to the source node.
- o It is susceptible to attack because ICMP messages are easily forged [RFC5927].

FOOTNOTE: According to RFC 0791, every IPv4 host must be capable of receiving a packet whose length is equal to 576 bytes. However, the IPv4 minimum link MTU is not 576. Section 3.2 of RFC 0791 explicitly states that the IPv4 minimum link MTU is 68 bytes.

FOOTNOTE: In the paragraphs above, the term "non-fragmentable packet" is introduced. A non-fragmentable packet can be fragmented at its source. However, it cannot be fragmented by a downstream node. An IPv4 packet whose DF-bit is set to zero is fragmentable. An IPv4 packet whose DF-bit is set to one is non-fragmentable. All IPv6 packets are also non-fragmentable.

FOOTNOTE: In the paragraphs above, the term "ICMP PTB message" is introduced. The ICMP PTB message has two instantiations. In ICMPv4 [RFC0792], the ICMP PTB message is Destination Unreachable message with Code equal to (4) fragmentation needed and DF set. This message was augmented by [RFC1191] to indicate the MTU of the link through which the packet could not be forwarded. In ICMPv6 [RFC4443], the ICMP PTB message is a Packet Too Big Message with Code equal to (0). This message also indicates the MTU of the link through which the packet could not be forwarded.

2.2. Upper-layer Protocols

When an upper-layer protocol submits data to the underlying IP module, and the resulting IP packet's length is greater than the PMTU, IP fragmentation may be required. IP fragmentation divides a packet into fragments. Each fragment includes an IP header and a portion of the original packet.

[RFC0791] describes IPv4 fragmentation procedures. IPv4 packets whose DF-bit is set to one cannot be fragmented. IPv4 packets whose DF-bit is set to zero can be fragmented at the source node or by any downstream router. [RFC8200] describes IPv6 fragmentation procedures. IPv6 packets can be fragmented at the source node only.

IPv4 fragmentation differs slightly from IPv6 fragmentation. However, in both IP versions, the upper-layer header appears in the first fragment only. It does not appear in subsequent fragments.

Upper-layer protocols can operate in the following modes:

- o Do not rely on IP fragmentation.
- o Rely on IP source fragmentation only (i.e., fragmentation at the source node).
- o Rely on IP source fragmentation and downstream fragmentation (i.e., fragmentation at any node along the path).

Upper-layer protocols running over IPv4 can operate in the first and third modes (above). Upper-layer protocols running over IPv6 can operate in the first and second modes (above).

Upper-layer protocols that operate in the first two modes (above) require access to the PMTU estimate. In order to fulfil this requirement, they can

- o Estimate the PMTU to be equal to the IPv4 or IPv6 minimum link MTU.
- o Access the estimate that PMTUD produced.
- o Execute PMTUD procedures themselves.
- o Execute Packetization Layer PMTUD (PLPMTUD) [RFC4821] [I-D.fairhurst-tsvwg-datagram-plpmtud] procedures.

According to PLPMTUD procedures, the upper-layer protocol maintains a running PMTU estimate. It does so by sending probe packets of various sizes to its peer and receiving acknowledgements. This strategy differs from PMTUD in that it relies on acknowledgement of received messages, as opposed to ICMP PTB messages concerning dropped messages. Therefore, PLPMTUD does not rely on the network's ability to deliver ICMP PTB messages to the source.

An upper-layer protocol that does not rely on IP fragmentation never causes the underlying IP module to emit

- o A fragmentable IP packet (i.e., an IPv4 packet with the DF-bit set to zero).
- o An IP fragment.
- o A packet whose length is greater than the PMTU estimate.

However, when the PMTU estimate is greater than the actual PMTU, the upper-layer protocol can cause the underlying IP module to emit a packet whose length is greater than the actual PMTU. When this occurs, a downstream router drops the packet and the source node refines its PMTU estimate, employing either PMTUD or PLPMTUD procedures.

When an upper-layer protocol that relies on IP source fragmentation only submits data to the underlying IP module, and the resulting packet is larger than the PMTU estimate, the underlying IP module fragments the packet and emits the fragments. However, the upper-layer protocol never causes the underlying IP module to emit

- o A fragmentable IP packet.
- o A packet whose length is greater than the PMTU estimate.

When the PMTU estimate is greater than the actual PMTU, the upper-layer protocol can cause the underlying IP module to emit a packet whose length is greater than the actual PMTU. When this occurs, a downstream router drops the packet and the source node refines its PMTU estimate, employing either PMTUD or PLPMTUD procedures.

An upper-layer protocol that relies on IP source fragmentation and downstream fragmentation can cause the underlying IP module to emit

- o A fragmentable IP packet.
- o An IP fragment.
- o A packet whose length is greater than the PMTU estimate.

A protocol that relies on IP source fragmentation and downstream fragmentation does not require access to the PMTU estimate. For these protocols, the underlying IP module:

- o Fragments all packets whose length exceeds the MTU of the first link along the path to the destination.
- o Sets the DF-bit to zero, so that downstream nodes can fragment the packet.

3. 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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

4. IP Fragmentation Reduces Reliability

This section explains how IP fragmentation reduces the reliability of Internet communication.

4.1. Middle Box Failures

Many middle boxes require access to the transport-layer header. However, when a packet is divided into fragments, the transport-layer header appears in the first fragment only. It does not appear in subsequent fragments. This omission can prevent middle boxes from delivering their intended services.

For example, assume that a router diverts selected packets from their normal path towards network appliances that support deep packet

inspection and lawful intercept. The router selects packets for diversion based upon the following 5-tuple:

- o IP Source Address.
- o IP Destination Address.
- o IPv4 Protocol or IPv6 Next Header.
- o transport-layer source port.
- o transport-layer destination port.

IP fragmentation causes this selection algorithm to behave suboptimally, because the transport-layer header appears only in the first fragment of each packet.

In another example, a middle box remarks a packet's Differentiated Services Code Point [RFC2474] based upon the above mentioned 5-tuple. IP fragmentation causes this process to behave suboptimally, because the transport-layer header appears only in the first fragment of each packet.

In all of the above-mentioned examples, the middle box cannot deliver its intended service without reassembling fragmented packets.

4.2. Partial Filtering

IP fragments cause problems for firewalls whose filter rules include decision making based on TCP and UDP ports. As the port information is not in the trailing fragments the firewall may elect to accept all trailing fragments, which may admit certain classes of attack, or may elect to block all trailing fragments, which may block otherwise legitimate traffic, or may elect to reassemble all fragmented packets, which may be inefficient and negatively affect performance.

4.3. Suboptimal Load Balancing

Many stateless load-balancers require access to the transport-layer header. Assume that a load-balancer distributes flows among parallel links. In order to optimize load balancing, the load-balancer sends every packet or packet fragment belonging to a flow through the same link.

In order to assign a packet or packet fragment to a link, the load-balancer executes an algorithm. If the packet or packet fragment contains a transport-layer header, the load balancing algorithm accepts the following 5-tuple as input:

- o IP Source Address.
- o IP Destination Address.
- o IPv4 Protocol or IPv6 Next Header.
- o transport-layer source port.
- o transport-layer destination port.

However, if the packet or packet fragment does not contain a transport-layer header, the load balancing algorithm accepts only the following 3-tuple as input:

- o IP Source Address.
- o IP Destination Address.
- o IPv4 Protocol or IPv6 Next Header.

Therefore, non-fragmented packets belonging to a flow can be assigned to one link while fragmented packets belonging to the same flow can be divided between that link and another. This can cause suboptimal load balancing.

4.4. Security Vulnerabilities

Security researchers have documented several attacks that rely on IP fragmentation. The following are examples:

- o Overlapping fragment attack [RFC1858] [RFC5722]
- o Resource exhaustion attacks (such as the Rose Attack)
- o Attacks based on predictable fragment Identification values [RFC7739]
- o Attacks based on bugs in the implementation of the fragment reassembly algorithm
- o Evasion of Network Intrusion Detection Systems (NIDS) [Ptacek1998]

In the overlapping fragment attack, an attacker constructs a series of packet fragments. The first fragment contains an IP header, a transport-layer header, and some transport-layer payload. This fragment complies with local security policy and is allowed to pass through a stateless firewall. A second fragment, having a non-zero offset, overlaps with the first fragment. The second fragment also

passes through the stateless firewall. When the packet is reassembled, the transport layer header from the first fragment is overwritten by data from the second fragment. The reassembled packet does not comply with local security policy. Had it traversed the firewall in one piece, the firewall would have rejected it.

A stateless firewall cannot protect against the overlapping fragment attack. However, destination nodes can protect against the overlapping fragment attack by implementing the reassembly procedures described in RFC 1858 and RFC 8200. These reassembly procedures detect the overlap and discard the packet.

The fragment reassembly algorithm is a stateful procedure for an otherwise stateless protocol. As such, it can be exploited for resource exhaustion attacks. An attacker can construct a series of fragmented packets, with one fragment missing from each packet such that the reassembly process cannot complete. Thus, this attack causes resource exhaustion on the destination node, possibly denying reassembly services to other flows. This type of attack can be mitigated by flushing fragment reassembly buffers when necessary, at the expense of possibly dropping legitimate fragments.

An IP fragment contains an "Identification" field that, together with the IP Source Address and Destination Address of a packet, identifies fragments that correspond to the same original datagram, such that they can be reassembled together by the receiving host. Many implementations have employed predictable values for the Identification field, thus making it easy for an attacker to forge malicious IP fragments that would cause the reassembly procedure for legitimate packets to fail.

Over the years multiple IPv4 and IPv6 implementations have been found to have flaws in their implementation of the IP fragment reassembly algorithm, typically resulting in buffer overflows. These buffer overflows have been exploitable for denial of service and remote code execution attacks.

NIDS aims at identifying malicious activity by analyzing network traffic. Ambiguity in the possible result of the fragment reassembly process may allow an attacker to evade these systems. Many of these systems try to mitigate some of these evasion techniques by e.g. computing all possible outcomes of the fragment reassembly process, at the expense of increased processing requirements.

4.5. Blackholing Due to ICMP Loss

As stated above, an upper-layer protocol requires access the PMTU estimate if it:

- o Does not rely on IP fragmentation.
- o Relies on IP source fragmentation only (i.e., fragmentation at the source node).

In order to satisfy this requirement, the upper-layer protocol can:

- o Estimate the PMTU to be equal to the IPv4 or IPv6 minimum link MTU.
- o Access the estimate that PMTUD produced.
- o Execute PMTUD procedures itself.
- o Execute PLPMTUD procedures.

PMTUD relies upon the network's ability to deliver ICMP PTB messages to the source node. Therefore, if an upper-layer protocol relies on PMTUD for its PMTU estimate, it also relies on the networks ability to deliver ICMP PTB messages to the source node.

[RFC4890] states that the PTB messages must not be filtered. However, ICMP delivery is not reliable. It is subject to transient loss and, in some configurations, more persistent delivery issues.

ICMP rate limiting, network congestion and packet corruption can cause transient loss. The effect of rate limiting may be severe, as RFC 4443 recommends strict rate limiting of IPv6 traffic.

While transient loss causes PMTUD to perform less efficiently, it does not cause PMTUD to fail completely. When the conditions contributing to transient loss abate, the network regains its ability to deliver ICMP PTB messages and PMTUD regains its ability to function.

By contrast, more persistent delivery issues cause PMTUD to fail completely. Consider the following example:

A DNS client sends a request to an anycast address. The network routes that DNS request to the nearest instance of that anycast address (i.e., a DNS Server). The DNS server generates a response and sends it back to the DNS client. While the response does not

exceed the DNS server's PMTU estimate, it does exceed the actual PMTU.

A downstream router drops the packet and sends an ICMP PTB message the packet's source (i.e., the anycast address). The network routes the ICMP PTB message to the anycast instance closest to the downstream router. Sadly, that anycast instance may not be the DNS server that originated the DNS response. It may be another DNS server with the same anycast address. The DNS server that originated the response may never receive the ICMP PTB message and may never update its PMTU estimate.

The problem described in this section is specific to PMTUD. It does not occur when the upper-layer protocol obtains its PMTU estimate from PLPMTUD or any other source.

Furthermore, the problem described in this section occurs when the upper-layer protocol does not rely on IP fragmentation, as well as when the upper-layer protocol relies on IP source fragmentation only.

4.6. Blackholing Due To Filtering

In RFC 7872, researchers sampled Internet paths to determine whether they would convey packets that contain IPv6 extension headers. Sampled paths terminated at popular Internet sites (e.g., popular web, mail and DNS servers).

The study revealed that at least 28% of the sampled paths did not convey packets containing the IPv6 Fragment extension header. In most cases, fragments were dropped in the destination autonomous system. In other cases, the fragments were dropped in transit autonomous systems.

Another recent study [Huston] confirmed this finding. It reported that 37% of sampled endpoints used IPv6-capable DNS resolvers that were incapable of receiving a fragmented IPv6 response.

It is difficult to determine why network operators drop fragments. In some cases, packet drop may be caused by misconfiguration. In other cases, network operators may consciously choose to drop IPv6 fragments, in order to address the issues raised in Section 4.1 through Section 4.5, above.

5. Alternatives to IP Fragmentation

5.1. Transport Layer Solutions

The Transport Control Protocol (TCP) [RFC0793]) can be operated in a mode that does not require IP fragmentation.

Applications submit a stream of data to TCP. TCP divides that stream of data into segments, with no segment exceeding the TCP Maximum Segment Size (MSS). Each segment is encapsulated in a TCP header and submitted to the underlying IP module. The underlying IP module prepends an IP header and forwards the resulting packet.

If the TCP MSS is sufficiently small, the underlying IP module never produces a packet whose length is greater than the actual PMTU. Therefore, IP fragmentation is not required.

TCP offers the following mechanisms for MSS management:

- o Manual configuration
- o PMTUD
- o PLPMTUD

For IPv6 nodes, manual configuration is always applicable. If the MSS is manually configured to 1220 bytes and the packet does not contain extension headers, the IP layer will never produce a packet whose length is greater than the IPv6 minimum link MTU (1280 bytes). However, manual configuration prevents TCP from taking advantage of larger link MTU's.

RFC 8200 strongly recommends that IPv6 nodes implement PMTUD, in order to discover and take advantage of path MTUs greater than 1280 bytes. However, as mentioned in Section 2.1, PMTUD relies upon the network's ability to deliver ICMP PTB messages. Therefore, PMTUD is applicable only in environments where the risk of ICMP PTB loss is acceptable.

By contrast, PLPMTUD does not rely upon the network's ability to deliver ICMP PTB messages. However, in many loss-based TCP congestion control algorithms, the dropping of a packet may cause the TCP control algorithm to drop the congestion control window, or even re-start with the entire slow start process. For high capacity, long RTT, large volume TCP streams, the deliberate probing with large packets and the consequent packet drop may impose too harsh a penalty on total TCP throughput for it to be a viable approach. [RFC4821] defines PLPMTUD procedures for TCP.

While TCP will never cause the underlying IP module to emit a packet that is larger than the PMTU estimate, it can cause the underlying IP module to emit a packet that is larger than the actual PMTU. If this occurs, the packet is dropped, the PMTU estimate is updated, the segment is divided into smaller segments and each smaller segment is submitted to the underlying IP module.

The Datagram Congestion Control Protocol (DCCP) [RFC4340] and the Stream Control Protocol (SCP) [RFC4960] also can be operated in a mode that does not require IP fragmentation. They both accept data from an application and divide that data into segments, with no segment exceeding a maximum size. Both DCCP and SCP offer manual configuration, PMTUD and PLPMTUD as mechanisms for managing that maximum size. [I-D.fairhurst-tsvwg-datagram-plpmtud] proposes PLPMTUD procedures for DCCP and SCP.

5.2. Application Layer Solutions

[RFC8085] recognizes that IP fragmentation reduces the reliability of Internet communication. Therefore, it offers the following advice regarding applications that run over the User Data Protocol (UDP) [RFC0768].

"An application SHOULD NOT send UDP datagrams that result in IP packets that exceed the Maximum Transmission Unit (MTU) along the path to the destination. Consequently, an application SHOULD either use the path MTU information provided by the IP layer or implement Path MTU Discovery (PMTUD) itself to determine whether the path to a destination will support its desired message size without fragmentation."

RFC 8085 continues:

"Applications that do not follow the recommendation to do PMTU/PLPMTUD discovery SHOULD still avoid sending UDP datagrams that would result in IP packets that exceed the path MTU. Because the actual path MTU is unknown, such applications SHOULD fall back to sending messages that are shorter than the default effective MTU for sending (EMTU_S in [RFC1122]). For IPv4, EMTU_S is the smaller of 576 bytes and the first-hop MTU. For IPv6, EMTU_S is 1280 bytes. The effective PMTU for a directly connected destination (with no routers on the path) is the configured interface MTU, which could be less than the maximum link payload size. Transmission of minimum-sized UDP datagrams is inefficient over paths that support a larger PMTU, which is a second reason to implement PMTU discovery."

RFC 8085 assumes that for IPv4, an EMTU_S of 576 is sufficiently small, even though the IPv4 minimum link MTU is 68 bytes.

This advice applies equally to application that run directly over IP.

6. Applications That Rely on IPv6 Fragmentation

The following applications rely on IPv6 fragmentation:

- o DNS [RFC1035]
- o OSPFv3 [RFC5340]
- o IP Encapsulations

Each of these applications relies on IPv6 fragmentation to a varying degree. In some cases, that reliance is essential, and cannot be broken without fundamentally changing the protocol. In other cases, that reliance is incidental, and most implementations already take appropriate steps to avoid fragmentation.

This list is not comprehensive, and other protocols that rely on IPv6 fragmentation may exist. They are not specifically considered in the context of this document.

6.1. DNS

DNS can obtain transport services from either UDP or TCP. Superior performance and scaling characteristics are observed when DNS runs over UDP.

DNS Servers that execute DNSSEC [RFC4035] procedures are more likely to generate large responses. Therefore, when running over UDP, they are more likely to cause the generation of IPv6 fragments. DNS's reliance upon IPv6 fragmentation is fundamental and cannot be broken without changing the DNS specification.

DNS is an essential part of the Internet architecture. Therefore, this issue is for further study and must be resolved before DNSSEC can be deployed successfully in IPv6 only networks.

6.2. OSPFv3

OSPFv3 implementations can emit messages large enough to cause IPv6 fragmentation. However, in keeping with the recommendations of RFC8200, and in order to optimize performance, most OSPFv3 implementations restrict their maximum message size to the IPv6 minimum link MTU.

6.3. IP Encapsulations

In this document, IP encapsulations include IP-in-IP [RFC2003], Generic Routing Encapsulation (GRE) [RFC2784], GRE-in-UDP [RFC8086] and Generic Packet Tunneling in IPv6 [RFC2473]. The fragmentation strategy described for GRE in [RFC7588] has been deployed for all of the above-mentioned IP encapsulations. This strategy does not rely on IPv6 fragmentation except in one corner case. (see Section 3.3.2.2 of RFC 7588 and Section 7.1 of RFC 2473). Section 3.3 of [RFC7676] further describes this corner case.

7. Recommendation

7.1. For Application Developers

Application developers SHOULD NOT develop applications that rely on IPv6 fragmentation.

Application-layer protocols then depend upon IPv6 fragmentation SHOULD be updated to break that dependency.

7.2. For Network Operators

As per RFC 4890, network operators MUST NOT filter ICMPv6 PTB messages unless they are known to be forged or otherwise illegitimate. As stated in Section 4.5, filtering ICMPv6 PTB packets causes PMTUD to fail. Many upper-layer protocols rely on PMTUD.

8. IANA Considerations

This document makes no request of IANA.

9. Security Considerations

This document mitigates some of the security considerations associated with IP fragmentation by discouraging the use of IP fragmentation. It does not introduce any new security vulnerabilities, because it does not introduce any new alternatives to IP fragmentation. Instead, it recommends well-understood alternatives.

10. Acknowledgements

TBD

11. References

11.1. Normative References

- [RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <<https://www.rfc-editor.org/info/rfc768>>.
- [RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <<https://www.rfc-editor.org/info/rfc791>>.
- [RFC0792] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, DOI 10.17487/RFC0792, September 1981, <<https://www.rfc-editor.org/info/rfc792>>.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<https://www.rfc-editor.org/info/rfc793>>.
- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <<https://www.rfc-editor.org/info/rfc1035>>.
- [RFC1191] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, DOI 10.17487/RFC1191, November 1990, <<https://www.rfc-editor.org/info/rfc1191>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4443] Conta, A., Deering, S., and M. Gupta, Ed., "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", STD 89, RFC 4443, DOI 10.17487/RFC4443, March 2006, <<https://www.rfc-editor.org/info/rfc4443>>.
- [RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, DOI 10.17487/RFC4821, March 2007, <<https://www.rfc-editor.org/info/rfc4821>>.
- [RFC8085] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, DOI 10.17487/RFC8085, March 2017, <<https://www.rfc-editor.org/info/rfc8085>>.

- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <<https://www.rfc-editor.org/info/rfc8200>>.
- [RFC8201] McCann, J., Deering, S., Mogul, J., and R. Hinden, Ed., "Path MTU Discovery for IP version 6", STD 87, RFC 8201, DOI 10.17487/RFC8201, July 2017, <<https://www.rfc-editor.org/info/rfc8201>>.

11.2. Informative References

- [Anderson2001] Anderson, J., "An Analysis of Fragmentation Attacks", 2001, <<http://www.ouah.org/fragma.html>>.
- [Huston] Huston, G., "IPv6, Large UDP Packets and the DNS (<http://www.potaroo.net/ispcol/2017-08/xtn-hdrs.html>)", August 2017.
- [I-D.fairhurst-tsvwg-datagram-plpmtud] Fairhurst, G., Jones, T., Tuexen, M., and I. Ruengeler, "Packetization Layer Path MTU Discovery for Datagram Transports", draft-fairhurst-tsvwg-datagram-plpmtud-02 (work in progress), December 2017.
- [Ptacek1998] Ptacek, T. and T. Newsham, "Insertion, Evasion and Denial of Service: Eluding Network Intrusion Detection", 1998, <<http://www.aciri.org/vern/Ptacek-Newsham-Evasion-98.ps>>.
- [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <<https://www.rfc-editor.org/info/rfc1122>>.
- [RFC1858] Ziemba, G., Reed, D., and P. Traina, "Security Considerations for IP Fragment Filtering", RFC 1858, DOI 10.17487/RFC1858, October 1995, <<https://www.rfc-editor.org/info/rfc1858>>.
- [RFC2003] Perkins, C., "IP Encapsulation within IP", RFC 2003, DOI 10.17487/RFC2003, October 1996, <<https://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, <<https://www.rfc-editor.org/info/rfc2473>>.
- [RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, DOI 10.17487/RFC2474, December 1998, <<https://www.rfc-editor.org/info/rfc2474>>.
- [RFC2784] Farinacci, D., Li, T., Hanks, S., Meyer, D., and P. Traina, "Generic Routing Encapsulation (GRE)", RFC 2784, DOI 10.17487/RFC2784, March 2000, <<https://www.rfc-editor.org/info/rfc2784>>.
- [RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, DOI 10.17487/RFC4035, March 2005, <<https://www.rfc-editor.org/info/rfc4035>>.
- [RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, DOI 10.17487/RFC4340, March 2006, <<https://www.rfc-editor.org/info/rfc4340>>.
- [RFC4890] Davies, E. and J. Mohacsi, "Recommendations for Filtering ICMPv6 Messages in Firewalls", RFC 4890, DOI 10.17487/RFC4890, May 2007, <<https://www.rfc-editor.org/info/rfc4890>>.
- [RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <<https://www.rfc-editor.org/info/rfc4960>>.
- [RFC5340] Coltun, R., Ferguson, D., Moy, J., and A. Lindem, "OSPF for IPv6", RFC 5340, DOI 10.17487/RFC5340, July 2008, <<https://www.rfc-editor.org/info/rfc5340>>.
- [RFC5722] Krishnan, S., "Handling of Overlapping IPv6 Fragments", RFC 5722, DOI 10.17487/RFC5722, December 2009, <<https://www.rfc-editor.org/info/rfc5722>>.
- [RFC5927] Gont, F., "ICMP Attacks against TCP", RFC 5927, DOI 10.17487/RFC5927, July 2010, <<https://www.rfc-editor.org/info/rfc5927>>.

- [RFC7588] Bonica, R., Pignataro, C., and J. Touch, "A Widely Deployed Solution to the Generic Routing Encapsulation (GRE) Fragmentation Problem", RFC 7588, DOI 10.17487/RFC7588, July 2015, <<https://www.rfc-editor.org/info/rfc7588>>.
- [RFC7676] Pignataro, C., Bonica, R., and S. Krishnan, "IPv6 Support for Generic Routing Encapsulation (GRE)", RFC 7676, DOI 10.17487/RFC7676, October 2015, <<https://www.rfc-editor.org/info/rfc7676>>.
- [RFC7739] Gont, F., "Security Implications of Predictable Fragment Identification Values", RFC 7739, DOI 10.17487/RFC7739, February 2016, <<https://www.rfc-editor.org/info/rfc7739>>.
- [RFC7872] Gont, F., Linkova, J., Chown, T., and W. Liu, "Observations on the Dropping of Packets with IPv6 Extension Headers in the Real World", RFC 7872, DOI 10.17487/RFC7872, June 2016, <<https://www.rfc-editor.org/info/rfc7872>>.
- [RFC8086] Yong, L., Ed., Crabbe, E., Xu, X., and T. Herbert, "GRE-in-UDP Encapsulation", RFC 8086, DOI 10.17487/RFC8086, March 2017, <<https://www.rfc-editor.org/info/rfc8086>>.

Appendix A. Contributors' Address

Authors' Addresses

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

Email: rbonica@juniper.net

Fred Baker
Unaffiliated
Santa Barbara, California 93117
USA

Email: FredBaker.IETF@gmail.com

Geoff Huston
APNIC
6 Cordelia St
Brisbane, 4101 QLD
Australia

Email: gih@apnic.net

Robert M. Hinden
Check Point Software
959 Skyway Road
San Carlos, California 94070
USA

Email: bob.hinden@gmail.com

Ole Troan
Cisco
Philip Pedersens vei 1
N-1366 Lysaker
Norway

Email: ot@cisco.com

Fernando Gont
SI6 Networks
Evaristo Carriego 2644
Haedo, Provincia de Buenos Aires
Argentina

Email: fgont@si6networks.com

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: July 7, 2018

D. O'Reilly
January 3, 2018

Approaches to Address the Availability of Information in Criminal
Investigations Involving Large-Scale IP Address Sharing Technologies
draft-daveor-cgn-logging-02

Abstract

The use of large-scale IP address sharing technologies (commonly known as "Carrier-Grade NAT" and "A+P") presents a challenge for law enforcement agencies due to the fact that incoming source port information is not routinely logged by Internet-facing servers. The absence of this information means that it is becoming increasingly difficult for law enforcement agencies to identify suspects in criminal activity online. This document considers the reasons why source port information is not routinely logged by Internet-facing servers and proposes some immediate-term actions that can be taken to help improve the situation. A deployment maturity model has been developed and a study of the support for logging incoming source port information in common server software is also presented.

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 <https://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 July 7, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
2. Scope	4
3. Centralised Connection Logging	5
4. Challenges to Capturing Source Port	7
4.1. Lack of Awareness	7
4.2. Lack of Support for Logging Source Port	8
4.3. Additional Storage Requirements	8
4.4. Default Log Formats	8
4.5. Breaking Existing Tooling	9
4.6. Accuracy of Recorded Time	9
4.7. Translation of Source Port by Intermediate Infrastructure	9
5. Comparison Model	10
6. Support for Logging Source Port	10
7. Conclusions and Next Steps	11
7.1. Raise Awareness of the Importance of Logging Source Port	12
7.2. Increase Support for Logging Source Port	12
7.3. Update Default Log Formats	12
7.4. Parallel Logging to a Connection Log	12
7.5. Adequate Timestamp Accuracy in Logs	13
7.6. Address Source Port Translation in Intermediate Infrastructure	13
8. IANA Considerations	14
9. Security Considerations	14
10. References	14
10.1. Informative References	14
10.2. Normative References	15
Appendix A. Support for Source Port Logging in Various Server Software	17
Author's Address	17

1. Introduction

Large-scale IP address sharing technologies (often collectively referred to as "Carrier-Grade NAT", [RFC6888]) are a helpful tool for extending the life of IPv4 addresses by allowing multiple endpoints to share a small number of IPv4 addresses. A number of such

technologies have been discussed and deployed, such as Dual-Stack Lite [RFC6333], NAT64 [RFC6146] and NAT444 [I-D.shirasaki-nat444]. A related category of technologies, known as "Address plus Port", or "A+P" [RFC6346], are also used for large-scale IP address sharing, achieved in these cases by using some of the port number bits for addressing purposes. Multiple examples of this category of technologies are also available, including Lightweight 4over6 [RFC7596], MAP-E [RFC7597] and MAP-T [RFC7599].

All of these technologies involve extending the space of available IPv4 addresses by mapping communication from multiple endpoints to a single, or small number of shared addresses, through the use of port numbers. The detail of how this is achieved in each technology varies, but the principle remains the same in all cases.

From the perspective of a server on the Internet, endpoint traffic that has passed through IP address sharing infrastructure appears to be originating from the IP address of the address sharing appliance. Common practice at the present time is for servers to log the connection time and source IP address of incoming connections. However, the IP address of the address sharing appliance is not sufficient to identify the true source of the traffic because potentially hundreds or thousands of individual endpoints were using that IP address at the same time. If the need arises during a criminal investigation to identify the source of a specific connection, the source port and exact connection time will also be required. Without this additional information it is highly unlikely that it will be possible for law enforcement authorities to progress their investigations.

Information is required from at least two sources to establish the link from the logs of an Internet-facing server to a specific subscriber endpoint:

1. The administrator of the Internet-facing server must have logged enough information to enable the operator of the IP address sharing infrastructure to isolate a specific subscriber endpoint.
2. The operator of the IP address sharing infrastructure must have logged sufficient information (for a sufficient length of time) to be able, when provided with adequate data by a law enforcement agency, to isolate the relevant subscriber endpoint.

The operators of large-scale IP address sharing infrastructure, typically Internet Service Providers, are usually required by law to maintain records of which endpoint was using a particular IP address and port at a particular time. The period of time for which these records must be retained is defined by national legislation.

Irrespective of whether (and for how long) these records are available, a starting point is needed to indicate to an investigating law enforcement agency that a particular endpoint was involved in a suspected criminal activity under investigation. Without such a starting point, it would be very difficult to progress the investigation even as far as engagement with the operator of the address sharing infrastructure. The records of Internet-facing servers are often a crucial source of this type of evidence.

It has been recognised for some time that IP address sharing presents a challenge to the ability to trace network use and abuse [RFC7620]. Further, it has also been recognised that this challenge is likely to become more severe and widespread with the increased use of large-scale address sharing [RFC6269]. More recently, Europol has highlighted the issue of large-scale IP address sharing as a threat to Internet governance [EUROPOL_IOCTA]. It is reported that the problem of crime attribution related to the use of carrier-grade NAT technologies is regularly encountered by 90% of respondents to a survey on the topic.

Address sharing, including large-scale address sharing, is required as long as the use of IPv4 continues. Full deployment of IPv6 has the potential to ultimately eliminate the current attribution issues arising from the use of large-scale address sharing technologies, although presumably new attribution challenges will arise in that scenario. Since it is impossible to anticipate if or when full migration to IPv6 will take place, it is prudent to consider the implications of the transitional technologies until the need for them has been eliminated.

2. Scope

Previous work has already suggested as best practice the logging by Internet-facing servers of source IP address, source port and exact connection time [RFC6302]. However, this continues to be exceptional, rather than routine, logging practice. The purpose of this document is to consider in more detail how it might be possible to bring about routine logging by Internet-facing servers of the information needed to re-establish the ability to trace network abuse for criminal investigative purposes. This document specifically does not address or consider the logging requirements of operators of large-scale address sharing infrastructure. Instead, the focus is on the logging considerations of operators of Internet-facing servers. The main contributions of this document are:

1. To consider the reasons why source port logging is not routinely carried out.

2. To identify some possible solutions and workarounds for the reasons that source port logging is not routinely carried out.
3. To examine the feasibility of source port logging from the perspective of software support for this feature.

Clearly no single solution will address the problem of crime attribution on the Internet. Load balancers, proxies and other network infrastructure may also, intentionally or as a side-effect, obfuscate the true source of Internet traffic and these problems will continue to exist with or without the presence of large-scale address sharing technologies (like Carrier-Grade NAT and A+P). Nevertheless, at the time of writing large-scale address sharing technologies present a significant challenge to crime attribution, as highlighted by Europol in the above referenced link, and this document attempts to consider the challenges specifically presented by that category of technologies.

The discussion begins by considering whether centralised connection logging is a viable solution to the problem of subscriber identification in criminal investigations. This is followed by an examination of the reasons why source port logging is not currently routinely carried out. A model has been developed for the comparison of the maturity of various server deployments to log source port and a study of common server software has been performed to assess the status of support for this functionality. Many, but not all, enterprise server solutions that were examined made the logging of source port either "Possible" or "Feasible", as defined in the maturity model. Only one type of server software examined made the logging of source port "Default".

3. Centralised Connection Logging

When large-scale IP address sharing technologies are used, source IP address is no longer a sufficient identifier of an individual subscriber. At a minimum, source port and accurate timestamp information are also required to distinguish between the potentially large number of individual users of a specific IP address at a particular time. [RFC6269] points out that there are two solutions to the question of how adequate information can be recorded to identify the parties to a particular connection. They are:

1. Operators of IP address sharing infrastructure log mappings between (source IP address, source port) combinations and their subscribers. Server operators log the IP address and source port of incoming connections. This is referred to as source port logging.

2. Instead of relying on server operators to log the source port of incoming connections, operators of IP address sharing infrastructure log all combinations of (external IP address, external port, destination IP address) for outgoing connections. This is referred to as connection logging. Server operators log the IP address and timestamp of incoming connections, which is the common current practice.

Two challenges to the use of connection logging by operators of IP address sharing infrastructure are also presented in RFC6269. Briefly:

- o The volumes of data involved make centralised recording of destination IP addresses infeasible.
- o Many individuals using the same IP address to access a popular destination (e.g. a popular website) might mean that it is not possible to distinguish between the activity of one subscriber and another, even if connection records are kept by the operator of the address sharing infrastructure.

The first issue raised is that the volumes of data involved make centralised recording of destination IP addresses infeasible. Whether destination IP addresses are recorded or not, the volume of logs generated by a large-scale IP address sharing infrastructure will be substantial, and some approaches have been proposed to address this hurdle and make central connection logging more feasible, such as deterministic allocation of ports [RFC6269],[RFC7422] or allocation of port ranges [RFC7768],[RFC6346]. While arguments of infeasibility are not arguments in principle why such logging cannot be done, the volumes of data involved in recording every single outgoing connection in a large Internet service provider represent legitimate technical, commercial and operational arguments for why it can not work in practice. Some representative figures for the scales of data involved can be found in [RFC7422], wherein it is estimated that the logging overhead would be of the order of 150MB per subscriber, per month. For a service provider with one million subscribers, this would produce a volume of logs (uncompressed) of the order of 150 terabytes per month. Aside from the technical overhead of storing such a volume of data, searching and locating relevant records over an extended, legally mandated retention period would also present a significant technical challenge.

The second point raised in [RFC6269] against connection logging by operators of IP address sharing infrastructure suggests that even if connection logs store all combinations of (timestamp, source IP, source port, destination IP), if this information is queried in the

absence of source port because source port has not been recorded by the destination IP, this would not be sufficient to distinguish the activity of one individual from another in cases where the destination IP is a popular one. This problem is further exacerbated in the case of protocols that make multiple connections per session (e.g. HTTP/HTTPS). The implication of this point is that connection logging, despite potential significant technical and operational overhead, cannot guarantee that the information retained is sufficient to identify an individual suspect, even when all required records are available.

Finally, the privacy concerns arising from connection logging in this scenario have been repeatedly raised [RFC6888] and [I-D.ietf-behave-ipfix-nat-logging].

In summary, it is certainly clear that operators of address sharing infrastructure need to retain records to enable the identification of suspects, and such records must consist of, at least, sufficient information to identify an individual subscriber when provided with a timestamp, source IP, source port and destination IP. However, there is no centralised solution available that removes the need for server operators to retain source port information.

4. Challenges to Capturing Source Port

It is relatively easy to articulate the reason why the operator of an Internet-facing server would wish to retain source port information for incoming connections. If the server operator (or the users that they serve) finds themselves the victim of a crime, it is preferable that all information that could be needed by the server operator to facilitate a criminal investigation is available. On the other hand, there are reasons why a server operator might not have the required source port information. This section enumerates the factors that could negatively influence both the ability and the inclination of server operators to capture and record source port information.

4.1. Lack of Awareness

Server operators are principally focussed on delivering the services for which they are operating their infrastructure. One of the main problems with the increasing use of IP address sharing technologies is the lack of awareness on the part of server operators that there are direct implications for them in case they should become the victim of a crime.

At the time of writing, a minimal amount of material is available online concerning this issue, even for those actively seeking to find out about source port logging. Where specific guidance or

information has been provided by vendors in relation to the configuration of source port logging, no explanation is provided for why this might be something that server operators might consider desirable. For example [MSDN_IIS_LOG].

There is, therefore, a considerable awareness gap between the importance of this issue for the purpose of investigating criminal activity online and the awareness of those who need to act in advance of any criminality taking place to ensure that the information needed to facilitate a future investigation is available.

4.2. Lack of Support for Logging Source Port

Before a server operator can decide to log source port information, the server software must support logging of the source port of incoming connections. Many, but not all major software distributions support the logging of the source port of incoming connections. Clearly lack of support in server software is a technical obstacle for a server operator to logging source port at the endpoint. It may still be possible to log source port at some location before the server endpoint (e.g. at a reverse proxy) but absence of support in server software will mean that endpoint logging will not be possible.

4.3. Additional Storage Requirements

In cases where it is possible to simply add source port to the list of fields recorded in log entries, the additional storage required to preserve source port data is minimal; in the region of six bytes per log entry (maximum of five ASCII digits for the source port plus an additional delimiter).

However, in some cases where software supports logging source port of incoming connections, it has been noted that this can only be achieved by enabling verbose or debug logging in the software. This would substantially (and unnecessarily) increase the size of logs produced by the server and would also, in all probability, reduce the production performance of the server. These factors would undoubtedly negatively influence the decision by a server operator to log incoming source port.

4.4. Default Log Formats

Many major software distributions provide default log formats in their configuration files. A review of the default log format of some common server software has been carried out and in only one case was it found that the source port of incoming connections is logged by any of the default log formats.

4.5. Breaking Existing Tooling

Much commercial and free log analysis software, by default, expects logs to be in a particular format. Consider, for example, the ubiquity of the Apache Common and Extended Log Formats. The software can usually be configured to parse arbitrary log formats, but this is additional configuration work for a server operator. For example: [ANALOG_LOG_CONFIG],[AWSTATS_LOG_CONFIG]. Without migration planning, a change to default log formats would most likely cause substantial disruption to a considerable amount of downstream processing of server log files. In addition to commercially available software, many administrators have developed or downloaded scripts that expect logs to be in a standard log format.

Therefore, log processing software, and in particular custom scripts, may break if default log formats change unexpectedly. At least, the tooling may need to be updated to correctly process the additional fields newly present in log file.

4.6. Accuracy of Recorded Time

As well as recording the IP address and source port of the connection, it is important to record the exact time of the connection. It has been suggested that there is a need for keeping the exact time against some sort of global standard (e.g. NTP) [RFC6302], however this may not be possible for practical, security or legacy reasons. In practice, it is usually not necessary to keep time against a global standard, as long as time is recorded consistently. The reason for this is that any time offset between the server and the time recorded in another organisation's records (running address sharing infrastructure) can be calculated and compensated for manually. Time offsets of this nature are commonly encountered and well understood in the digital forensics world.

4.7. Translation of Source Port by Intermediate Infrastructure

It is common for an incoming connection to terminate somewhere other than the actual server that is intended to ultimately handle the connection. For example, it is possible that a server operator has deployed intermediate infrastructure to improve the efficiency or availability of their platform. Load balancers, proxies or denial of service countermeasures may be present, any one of which could potentially terminate the incoming connection. The operation of these types of intermediate infrastructure can cause translation of the incoming connection parameters (including source port) before the connection is established to the actual server endpoint.

In such cases the source port presented at the server endpoint is a source port that only has meaning in the intermediate infrastructure and in most cases will not carry any information about the source port in use at the connection origin. In the worst case scenario (from the point of view of crime attribution), the intermediate infrastructure may obfuscate the true source connection information in a way that is unrecoverable.

5. Comparison Model

A model has been developed to assist with comparison of the maturity of server software deployments to store and retrieve source port information for incoming connections. The model is depicted in Figure 1.

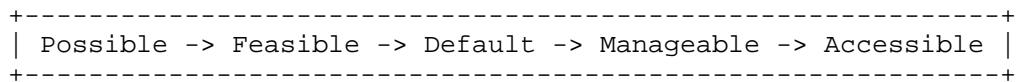


Figure 1

- o "Possible": Means that the server software supports, in any way, the ability to record source ports for incoming connections.
- o "Feasible": Means that it there are no significant performance or storage implications for enabling the storage of source ports.
- o "Default": Means that, at a minimum, at least one of the default log formats provided with the software distribution enables the storage of source ports.
- o "Manageable": Means that tooling is, or has been, build or adapted to support the storage of source ports.
- o "Accessible": Means that it is possible to identify and retrieve relevant records in the stored log data.

6. Support for Logging Source Port

Open-source research has been conducted to assess the status of support for logging of source port information in common server software.

The assessment criteria were as follows:

- o Server software is categorised as "Possible" if there was any way identified to cause the logging of source port.

- o Server software is categorised as "Feasible" if the logging of source port does not require increasing the log level to cause the logging of source port to be possible. In other words, if a server requires enabling verbose, debug or audit logging in order to be able to record source port then logging is "Possible" but not "Feasible".
- o Server software is categorised as "Default" if at least one of the available default log formats enables logging of the incoming source port, or if source port is logged by default.
- o The "Manageable" and "Accessible" aspects of the comparison model relate to specific deployments and are therefore not considered in the assessment of server software support.

The latest versions of 16 common server software packages have been examined and documentation has been research to identify if and how source port logging can be enabled. The findings are described in Appendix A. Online documentation has been examined to identify if and how source port logging can be enabled. The results are presented in the following table:

Possible	Feasible	Default	Manageable	Accessible
13	11	1	N/A	N/A

Table 1: Support Table

It was noted that only one of the server software packages examined (OpenSSH version 7.5) enables the logging of incoming source port by default. This conclusion has been reached despite using the most generous possible interpretation of "Default", whereby meeting the criteria for "Default" is achieved when logging of source port is offered as a possible default, rather than requiring that logging of source port is enabled by default. In due course, as awareness of this issue increases, it is envisioned that a stricter interpretation of "Default" would be more appropriate, requiring that the logging of source port be enabled by default.

7. Conclusions and Next Steps

There is clearly substantial work to be done to bring about the regular recording of source port information at Internet-facing servers and there are undoubtedly criminals free right now because the information required to identify them from their online activity is not available.

The next steps presented below are some possible courses of action that have been identified based on the current state of source port logging and the challenges described above.

7.1. Raise Awareness of the Importance of Logging Source Port

Publishers of both free and commercial software should consider releasing deployment guidance or best practice that describes why server administrators need to be recording source port information, with instructions for how this can be done. This will help to address the lack of awareness of the importance of this issue.

Considering also the awareness of those who are building software applications, or otherwise involved with coding of Internet-facing applications, secure coding guidance should be updated to include reference to source port information, particularly where such guidance already touches on the issue of logging. For example the OWASP Secure Coding Practices specifies a list of important log event data [OWASP_SCP]. However the "important log event data" list does not, at the time of writing, include source port.

7.2. Increase Support for Logging Source Port

Many software packages support logging of source port information, but only ten out of the sixteen examined support logging in a way that would not significantly negatively impact the operation of the server software. Software publishers therefore need to consider their level of support of logging source port. In particular, software should support the logging of source port without needing to enable a verbose logging level.

7.3. Update Default Log Formats

In cases where a particular software package has support for logging of incoming source port, one possibility would be to incorporate one or more log formats that include incoming source port as a field logged by default. Obviously this will not have any impact on deployments of the software that are already in place but for future deployments, the incorporation of source port into the log format will mean that those administrators that use the unaltered default log format will automatically store the required information.

7.4. Parallel Logging to a Connection Log

Where possible, configuring parallel logging of connection information to a separate log stream would be one possible solution to address the fact that changes to log format might break downstream tooling. This would also be a possible solution that could be used

by those server software types that log via syslog. In this case, software publishers could produce guidance on how to configure syslog to log connection information parallel to main log files.

Such a solution would help to ease the transition to an alternate log format since current log formats would not need to be changed because the required source port information is stored separately, but can still be correlated with the main log files if needed.

7.5. Adequate Timestamp Accuracy in Logs

Operators of large-scale address sharing infrastructure will, most likely need connection times specified with at least the granularity of a second. Most, but not all, server software will log times with this granularity by default but there is no guarantee that this is the case.

Consideration should be given by server operators to making sure that the times that are being recorded in their log files have sufficient accuracy to allow identification of the required records. As mentioned earlier, the times do not necessarily need to be recorded with reference to a centralised time source (e.g. NTP) as long as times are recorded consistently.

This factor also needs to be considered by software developers when they are producing software and although the recording of time is mentioned in the OWASP Secure Coding Practices, the required accuracy/granularity of the recorded time is not discussed [OWASP_SCP].

7.6. Address Source Port Translation in Intermediate Infrastructure

In cases described above where intermediate infrastructure terminates incoming connections (proxies, load balancers, etc.), and the infrastructure is translating incoming source port information, there is a risk that the important crime attribution information may be lost. One possibility is to log source port information at the intermediate infrastructure and this may be an appropriate solution in some cases. The problem is that this may lead to an excessive volume of logging, depending on the particular scenario. For example if the intermediate infrastructure is being used to mitigate DDoS attacks, logging all incoming traffic would potentially lead to logging of all incoming DDoS connections. This would clearly be an undesirable outcome.

An alternative solution is to pass information about the original connection (before mapping/translation of connection information takes place) to the actual endpoint. Solutions to achieve this

already exist for certain application layer protocols. The Forwarded HTTP Extension [RFC7239], for example, supports (as an optional feature) the transfer of source port information in the "Forwarded For" header, and this technique can also support multiple layers of proxying without loss of attribution.

8. IANA Considerations

This memo includes no request to IANA.

9. Security Considerations

Clearly a balance needs to be struck between individual right to privacy and law enforcement access to data during criminal investigations. On the one hand, the routine logging of any additional information has the potential to introduce risks related to privacy and human rights. On the other hand, it is fair to say that there are criminals free today because the data required to identify them is not available due to the use of large-scale address sharing technologies. Across the world there are also a broad spectrum of legislative regimes and human rights challenges, interpretation of which relate directly to this question.

IP addresses are routinely logged today and this information can be used for identification of people online in some cases. The cases in which an IP address does not identify an individual directly are not necessarily apparent to the person performing the logging (who cannot tell, for example, if the true source of the traffic is behind a NAT or other form of proxy) and the same is true even if source port is logged. It is not apparent that there is any additional risk to individual privacy between the case when a single piece of endpoint identifying information (source IP address) is logged versus the case when two pieces of endpoint identifying information (source IP address and source port) are logged. Balancing this against the significant advantages from the crime attribution point of view suggests that this may be a worthwhile approach.

10. References

10.1. Informative References

- [I-D.ietf-behave-ipfix-nat-logging]
Sivakumar, S. and R. Penno, "IPFIX Information Elements for logging NAT Events", draft-ietf-behave-ipfix-nat-logging-13 (work in progress), January 2017.

[I-D.shirasaki-nat444]
Yamagata, I., Shirasaki, Y., Nakagawa, A., Yamaguchi, J.,
and H. Ashida, "NAT444", draft-shirasaki-nat444-06 (work
in progress), July 2012.

10.2. Normative References

- [ANALOG_LOG_CONFIG]
Analog, "Analog 6.0: Log formats", 2017,
<<http://mirror.reverse.net/pub/analog/docs/logfmt.html>>.
- [AWSTATS_LOG_CONFIG]
AWStats, "AWStats Installation, Configuration and
Reporting (for version 7.6)", 2017,
<https://awstats.sourceforge.io/docs/awstats_setup.html>.
- [EUROPOL_IOCTA]
Europol, "The Internet Organised Crime Threat Assessment",
2016, <[https://www.europol.europa.eu/activities-services/
main-reports/
internet-organised-crime-threat-assessment-iocta-2016](https://www.europol.europa.eu/activities-services/main-reports/internet-organised-crime-threat-assessment-iocta-2016)>.
- [MSDN_IIS_LOG]
Microsoft, "IIS 8.5 - How to log client port number",
2015, <[https://blogs.msdn.microsoft.com/amb/2015/11/12/
iis-8-5-how-to-log-client-port-number/](https://blogs.msdn.microsoft.com/amb/2015/11/12/iis-8-5-how-to-log-client-port-number/)>.
- [OWASP_SCP]
OWASP, "OWASP Secure Coding Practices Quick Reference
Guide", 2010, <[https://www.owasp.org/images/0/08/
OWASP_SCP_Quick_Reference_Guide_v2.pdf](https://www.owasp.org/images/0/08/OWASP_SCP_Quick_Reference_Guide_v2.pdf)>.
- [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, <<https://www.rfc-editor.org/info/rfc6146>>.
- [RFC6269] Ford, M., Ed., Boucadair, M., Durand, A., Levis, P., and
P. Roberts, "Issues with IP Address Sharing", RFC 6269,
DOI 10.17487/RFC6269, June 2011,
<<https://www.rfc-editor.org/info/rfc6269>>.
- [RFC6302] Durand, A., Gashinsky, I., Lee, D., and S. Sheppard,
"Logging Recommendations for Internet-Facing Servers",
BCP 162, RFC 6302, DOI 10.17487/RFC6302, June 2011,
<<https://www.rfc-editor.org/info/rfc6302>>.

- [RFC6333] Durand, A., Droms, R., Woodyatt, J., and Y. Lee, "Dual-Stack Lite Broadband Deployments Following IPv4 Exhaustion", RFC 6333, DOI 10.17487/RFC6333, August 2011, <<https://www.rfc-editor.org/info/rfc6333>>.
- [RFC6346] Bush, R., Ed., "The Address plus Port (A+P) Approach to the IPv4 Address Shortage", RFC 6346, DOI 10.17487/RFC6346, August 2011, <<https://www.rfc-editor.org/info/rfc6346>>.
- [RFC6888] Perreault, S., Ed., Yamagata, I., Miyakawa, S., Nakagawa, A., and H. Ashida, "Common Requirements for Carrier-Grade NATs (CGNs)", BCP 127, RFC 6888, DOI 10.17487/RFC6888, April 2013, <<https://www.rfc-editor.org/info/rfc6888>>.
- [RFC7239] Petersson, A. and M. Nilsson, "Forwarded HTTP Extension", RFC 7239, DOI 10.17487/RFC7239, June 2014, <<https://www.rfc-editor.org/info/rfc7239>>.
- [RFC7422] Donley, C., Grundemann, C., Sarawat, V., Sundaresan, K., and O. Vautrin, "Deterministic Address Mapping to Reduce Logging in Carrier-Grade NAT Deployments", RFC 7422, DOI 10.17487/RFC7422, December 2014, <<https://www.rfc-editor.org/info/rfc7422>>.
- [RFC7596] Cui, Y., Sun, Q., Boucadair, M., Tsou, T., Lee, Y., and I. Farrer, "Lightweight 4over6: An Extension to the Dual-Stack Lite Architecture", RFC 7596, DOI 10.17487/RFC7596, July 2015, <<https://www.rfc-editor.org/info/rfc7596>>.
- [RFC7597] Troan, O., Ed., Dec, W., Li, X., Bao, C., Matsushima, S., Murakami, T., and T. Taylor, Ed., "Mapping of Address and Port with Encapsulation (MAP-E)", RFC 7597, DOI 10.17487/RFC7597, July 2015, <<https://www.rfc-editor.org/info/rfc7597>>.
- [RFC7599] Li, X., Bao, C., Dec, W., Ed., Troan, O., Matsushima, S., and T. Murakami, "Mapping of Address and Port using Translation (MAP-T)", RFC 7599, DOI 10.17487/RFC7599, July 2015, <<https://www.rfc-editor.org/info/rfc7599>>.
- [RFC7620] Boucadair, M., Ed., Chatras, B., Reddy, T., Williams, B., and B. Sarikaya, "Scenarios with Host Identification Complications", RFC 7620, DOI 10.17487/RFC7620, August 2015, <<https://www.rfc-editor.org/info/rfc7620>>.

[RFC7768] Tsou, T., Li, W., Taylor, T., and J. Huang, "Port Management to Reduce Logging in Large-Scale NATs", RFC 7768, DOI 10.17487/RFC7768, January 2016, <<https://www.rfc-editor.org/info/rfc7768>>.

Appendix A. Support for Source Port Logging in Various Server Software

The table below enumerates the findings of best-effort, open-source review of documentation of the various products. Where it has been indicated that it is not possible to log source port then either (a) no reference has been identified in online documentation to indicate how source port logging can be enabled, or (b) a reference positively indicating that logging of source port is not possible has been found.

Category	Server	Version	Possible	Feasible	Default
HTTP	Apache HTTPD	2.4.25	Yes	Yes	No
HTTP	IIS	10	Yes	Yes	No
HTTP	Tomcat	8.5.15	Yes	Yes	No
HTTP	Squid	3.5.25	Yes	Yes	No
HTTP	nginx	1.12.0	Yes	Yes	No
Mail	sendmail	8.15.2	Yes	Yes	No
Mail	Microsoft Exchange Server	2016	Yes	No	No
Mail	Postfix	2.10.0	Yes	Yes	No
Mail	Exim	4.89	Yes	Yes	No
Mail	Dovecot	2.2.30.1	Yes	Yes	No
Mail	UW IMAP	imap-2007f	No	No	No
DBase	Oracle	12.2.0.1	No	No	No
DBase	MySQL	5.7.18	No	No	No
DBase	Microsoft SQL Server	2016	Yes	No	No
DBase	PostgreSQL	9.6.3	Yes	Yes	No
SSH	OpenSSH	7.5	Yes	Yes	Yes

Table 2: Support for Logging Incoming Source Port

Author's Address

David O'Reilly
Ireland

Email: rfc@daveor.com

INTERNET-DRAFT
Intended Status: Informational
Expires: April 22, 2018

Tom Herbert
Quantonium
Petr Lapukhov
Facebook
October 19, 2017

Identifier-locator addressing for IPv6
draft-herbert-intarea-ila-00

Abstract

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

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/lid-abstracts.html>

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

Copyright and License Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1	Introduction	4
1.1	Terminology	4
1.2	Use cases	6
2	Architectural overview	6
2.1	Addressing	6
2.2	Network topology	7
2.3	Translations and mappings	8
2.4	ILA routing	8
2.5	ILA domains	9
2.6	ILA control plane	9
3	Address formats	10
3.1	ILA address format	10
3.2	Locators	10
3.3	Identifiers	10
3.4	Standard identifier representation addresses	11
4	Optional identifier formats	12
4.1	Checksum neutral mapping	12
4.2	Identifier types	12
4.2.1	Interface identifiers	15
4.2.2	Locally unique identifiers	15
4.2.3	Virtual networking identifiers for IPv4	15
4.2.4	Virtual networking identifiers for IPv6 unicast	16
4.2.5	Virtual networking identifiers for IPv6 multicast	17
4.2.6	Non-local address identifiers	18
4.3	SIR addresses with formatted identifiers	19
4.3.1	SIR for locally unique identifiers	19
4.3.2	SIR for virtual addresses	20
4.3.2	SIR for non-local address identifiers	20
5	Operation	20
5.1	Identifier to locator mapping	20
5.2	Address translations	21
5.2.1	SIR to ILA address translation	21
5.2.2	ILA to SIR address translation	21
5.3	Virtual networking operation	22

5.3.1	Crossing virtual networks	22
5.3.2	IPv4/IPv6 protocol translation	22
5.4	Transport layer checksums	22
5.4.1	Checksum-neutral mapping	23
5.4.2	Sending an unmodified checksum	25
5.5	Non-local address mapping	25
5.6	Address selection	26
5.7	Duplicate identifier detection	26
5.8	ICMP error handling	26
5.8.1	Handling ICMP errors by ILA capable hosts	26
5.8.2	Handling ICMP errors by non-ILA capable hosts	27
5.9	Multicast	27
6	Motivation for ILA	28
6.1	Use cases	28
6.1.1	Multi-tenant virtualization	28
6.1.2	Datacenter virtualization	28
6.1.3	Mobile networks	29
6.2	Alternative methods	29
6.2.1	ILNP	29
6.2.2	Flow label as virtual network identifier	30
6.2.3	Extension headers	30
6.2.4	Encapsulation techniques	31
7	IANA Considerations	31
8	References	32
8.1	Normative References	32
8.2	Informative References	32
9	Acknowledgments	33
	Appendix A: Communication scenarios	34
A.1	Terminology for scenario descriptions	34
A.2	Identifier objects	35
A.3	Reference network for scenarios	35
A.4	Scenario 1: Object to task	36
A.5	Scenario 2: Object to Internet	36
A.6	Scenario 3: Internet to object	36
A.7	Scenario 4: Tenant system to service	37
A.8	Scenario 5: Object to tenant system	37
A.9	Scenario 6: Tenant system to Internet	38
A.10	Scenario 7: Internet to tenant system	38
A.11	Scenario 8: IPv4 tenant system to object	38
A.12	Tenant to tenant system in the same virtual network	39
A.12.1	Scenario 9: TS to TS in the same VN using IPV6	39
A.12.2	Scenario 10: TS to TS in same VN using IPv4	39
A.13	Tenant system to tenant system in different virtual networks	39
A.13.1	Scenario 11: TS to TS in different VNs using IPV6	39
A.13.2	Scenario 12: TS to TS in different VNs using IPv4	40
A.13.3	Scenario 13: IPv4 TS to IPv6 TS in different VNs	40
A.14	Scenario 14: Non-local address to tenant system	40

Appendix B: unique identifier generation	41
B.1 Globally unique identifiers method	41
B.2 Universally Unique Identifiers method	42
Appendix C: Datacenter task virtualization	42
C.1 Address per task	42
C.2 Job scheduling	43
C.3 Task migration	43
C.3.1 Address migration	44
C.3.2 Connection migration	44
Appendix D: Mobility in wireless networks	45

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 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.

Some 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 6 provides a motivation for ILA and comparison of ILA with alternative methods that achieve similar functionality.

1.1 Terminology

ILA Identifier-locator addressing.

ILA host	An end host that is capable of performing ILA translations on transmit or receive.
ILA router	A network node that performs ILA translation and forwarding of translated packets.
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. 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.
Identifier address	An IP address that contains an identifier.
ILA address	An IPv6 address composed of a locator (upper sixty-four bits) and an identifier (low order sixty-four bits).
ILA domain	A unique identifier namespace indicated by a SIR prefix. Each SIR prefix maps to ILA 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.
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.
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.

Checksum-neutral mapping

A method to preserve a correct transport layer checksum when performing ILA translation. When the upper sixty-four bits of an address are overwritten in an ILA translation, a modification can be made to the low order bits of the identifier to offset the checksum difference.

1.2 Use cases

ILA use cases include datacenter virtualization, network virtualization, and mobility in cellular and other mobile networks. Section 6 provides details on these use cases. ILA operates at the network layer so it works with any transport layer protocol and can be used at intermediate devices or end nodes. An ILA implementation may include optimizations depending on where in the network it runs.

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 a 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 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.

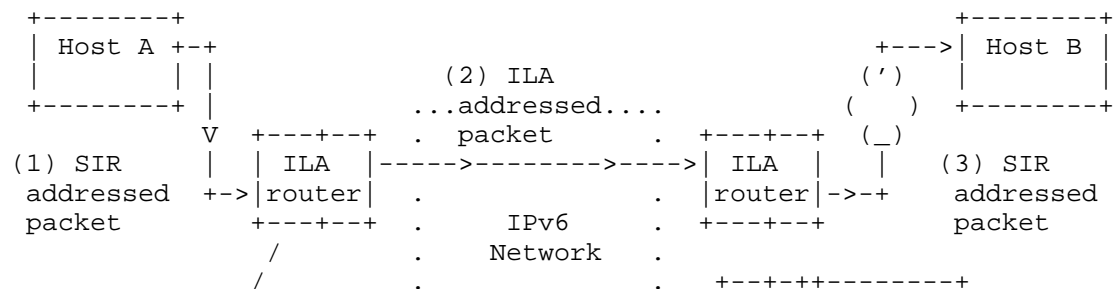
ILA translations can transform addresses from one type to another. In network virtualization, virtual addresses can be translated into ILA and SIR addresses, and conversely ILA and SIR addresses can be translated to virtual addresses.

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, an ILA node (host or router) will translate the virtual or SIR address of a destination to an ILA address. At 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. The locator is set to a value that will route packets to a peer ILA node that Host B is downstream of. The packet is forwarded over the network and delivered to the 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

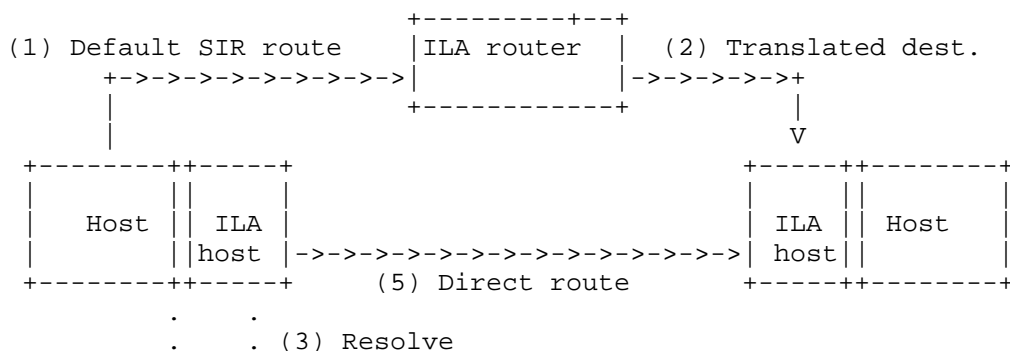
Address translation is the mechanism employed by ILA. Logical or virtual addresses are translated to topological IPv6 addresses for transport to the proper destination. Translation occurs in the upper sixty-four bits of an address, the low order sixty-four bits contains an identifier that is immutable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

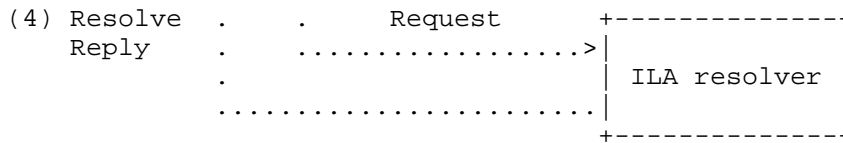
Each ILA node maintains a mapping table. This table maps identifiers to locators. The mappings are dynamic as nodes with identifiers can be created, destroyed, or move in the network. Mappings are propagated amongst ILA routers or hosts in a network using mapping propagation protocols (mapping propagation 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 device mobility. 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 location, 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 a 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 resolve 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.

2.5 ILA domains

An ILA domain defines a namespace for identifiers. Identifiers must be unique within an ILA domain. Each SIR prefix maps to one ILA domain so that the combination of a SIR prefix and an identifier (a SIR address) uniquely identifies a node. More than one SIR prefix may be associated with a domain in which case SIR addresses with different SIR prefixes but the same identifier would refer to the same node.

Locators MUST map to only one SIR domain in order to ensure that translation from a locator to SIR prefix is unambiguous.

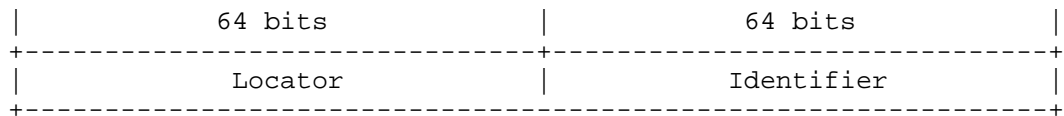
2.6 ILA control plane

ILA routers and ILA hosts assume a control plane that propagates the tables that map SIR addresses to ILA address (or just identifier to locator mappings). There are several possible methods for control planes that have been proposed including synchronized configuration, BGP, DNS, and NoSQL databases. Defining a specific control plane for ILA is out of scope of this document.

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]).

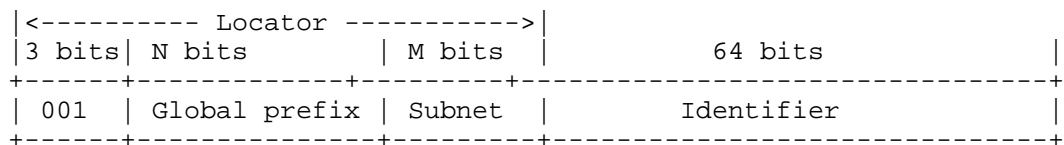


Note that there is no technical reason why identifiers and locators must be sixty-four bits. Different sizes could be used. The split is somewhat arbitrary, however it does simplify the description and implementation. For instance, sixty-four bits is the size of a "long long" native data type in several computer architectures. It is conceivable that a different arrangement could be used for some ILA domain. However, for the purposes of this document we assume the 64/64 split.

3.2 Locators

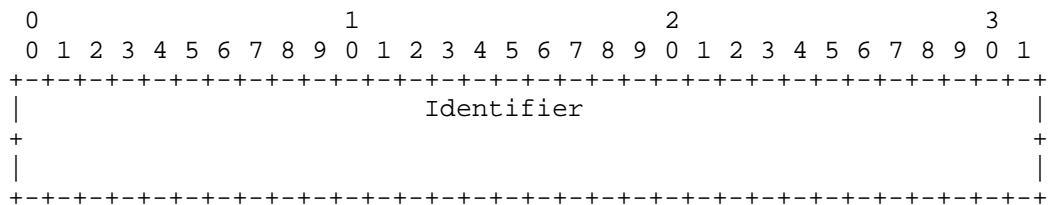
Locators are routable network address prefixes that create topological addresses for physical hosts within the network. They SHOULD be assigned from a global address block [RFC3587].

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



3.3 Identifiers

Identifiers uniquely identify logical nodes in an ILA domain. The format of an ILA identifier is:



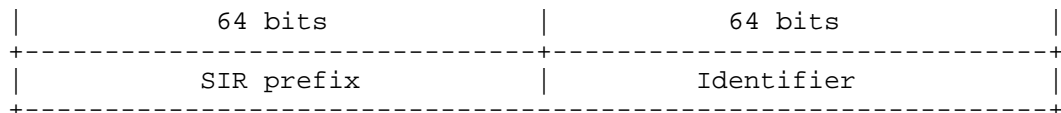
Identifiers are specified to be sixty-four bit values that are unstructured. A structure and format for identifiers MAY be defined for a domain; for instance the operator of an ILA domain may define the use of prefixes for its identifiers in order to facilitate hierarchies of its identifiers. Section 4 defines optional ILA formats that an ILA domain might impose locally that allow different types of identifiers as well as an indication of checksum neutral mapping.

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 an identifier to indicate the peer endpoint of the communication. The endpoints of an established connection context are also referenced by identifiers (encoded in SIR addresses). 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:



A SIR prefix SHOULD be a globally routable prefix per [RFC3587]. A globally routable SIR prefix facilitates connectivity between hosts on the Internet and ILA nodes. An ILA router between a site's network and the Internet can translate between SIR prefix and locator for an identifier. A network may have multiple SIR prefixes where each prefix defines a unique identifier space.

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

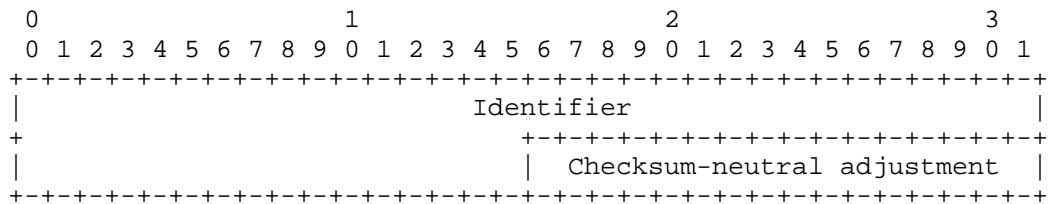
A standard identifier representation address can be used as the externally visible address for a node. This can be used throughout the network, returned in DNS AAAA records [RFC3363], used in logging, etc. An application can use a SIR address without knowledge that it encodes an identifier.

4 Optional identifier formats

This section describes optional identifier formats that allow for different types of identifiers, groups of identifiers, and checksum neutral mapping being applied. Note that identifiers are defined as unstructured fields, there is no required structure imposed on them. An administrator MAY impose an identifier format within an ILA domain. Any imposed structure is local only to the domain and all ILA nodes within the domain must agree on the format. A format might include optional elements as described below, or may include other elements customized for a domain.

4.1 Checksum neutral mapping

Checksum neutral mapping is an optional mechanism that may be applied to an ILA address (see section 5.4.1 for description of checksum-neutral mapping). When employed the checksum neutral mapping occupies the low order sixteen bits of the identifier in a locator address.



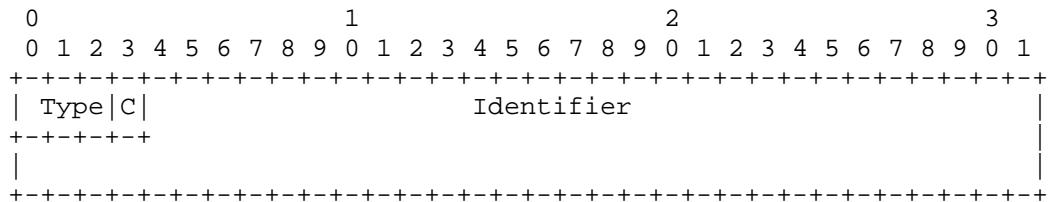
The presence of the checksum-neutral adjustment field must be unambiguous. An optional C-bit flag could be used in the identifier to indicate the checksum-neutral field is valid. The use of the C-bit is demonstrated below. Alternatively, within an ILA domain an operator could require it to be assumed that all ILA addresses have the checksum-neutral field set so that an explicit flag is not needed. Note that checksum-neutral adjustment is not used with SIR addresses.

4.2 Identifier types

This section describes an optional identifier format that allows for different types of identifiers and an indication of checksum neutral mapping being applied.

Note that the identifier type format is optional. If this is not used within an ILA domain then all ILA nodes assume that all identifiers are of the same type (locally unique identifier for instance).

The optional type format of an ILA identifier with the checksum adjust flag is:



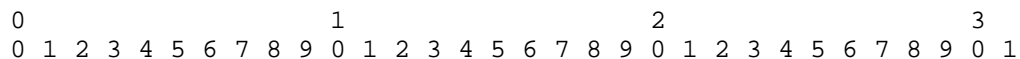
Fields are:

- o Type: Type of the identifier (see below).
- o C: The C-bit. This indicates that checksum-neutral mapping applied (see below). Presence of this field is optional.
- o Identifier: Identifier value.

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: non-local address identifier
- 6-7: Reserved

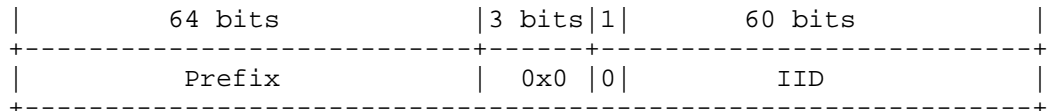
If the C-bit is set then the low order sixteen bits of an identifier contain the adjustment for checksum-neutral mapping (see section 4.4.1 for description of checksum-neutral mapping). The format of an identifier with checksum neutral mapping is:



```
+-----+
| Type|1|                               Identifier                               |
+-----+                               +-----+
|                               | Checksum-neutral adjustment |
+-----+
```

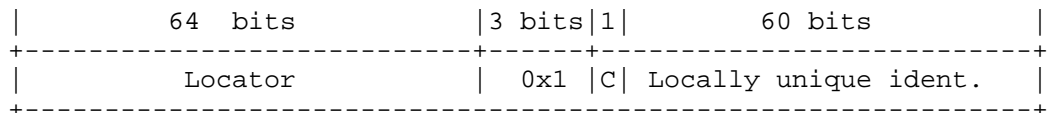
4.2.1 Interface identifiers

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:

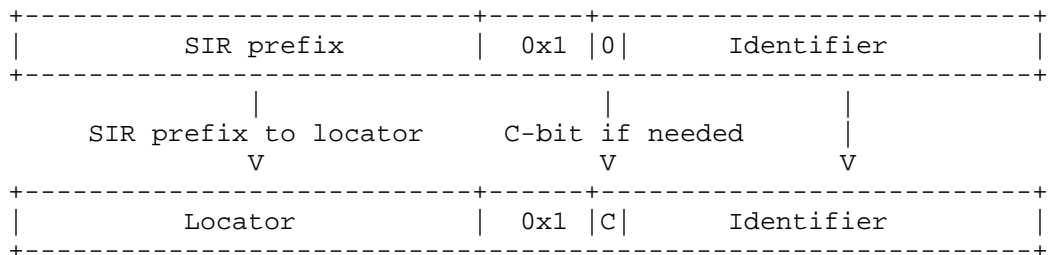


4.2.2 Locally unique identifiers

Locally unique identifiers (LUI) can be created for various addressable objects within a network. These identifiers are in a flat 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:



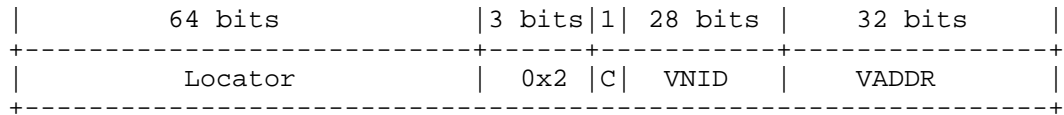
The figure below illustrates the translation from SIR address to an ILA address as would be performed when a node sends to a SIR address. Note the low order 16 bits of the identifier may be modified as the checksum-neutral adjustment. The reverse translation of ILA address to SIR address is symmetric.



4.2.3 Virtual networking identifiers for IPv4

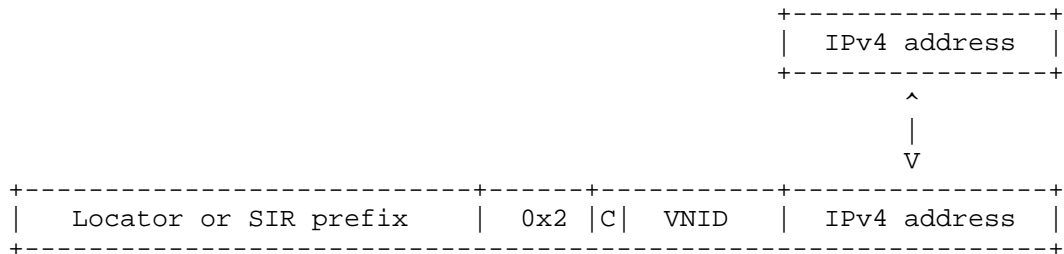
This type defines a format for encoding an IPv4 virtual address and virtual network identifier within an identifier. The format of an ILA

address for IPv4 virtual networking is:



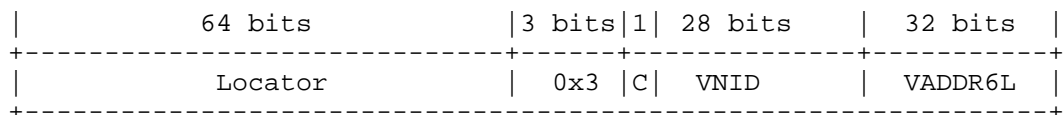
VNID is a virtual network identifier and VADDR is a virtual address within the virtual network indicated by the VNID. The VADDR can be an IPv4 unicast or multicast address, and may often be in a private address space (i.e. [RFC1918]) used in the virtual network.

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 between IPv4 and IPv6.



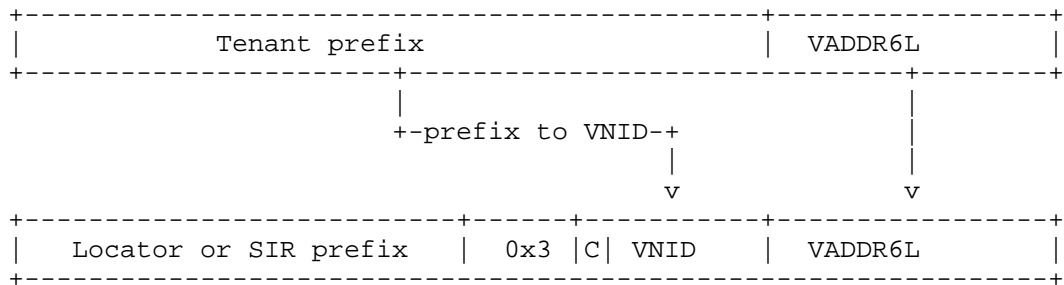
4.2.4 Virtual networking identifiers for IPv6 unicast

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:

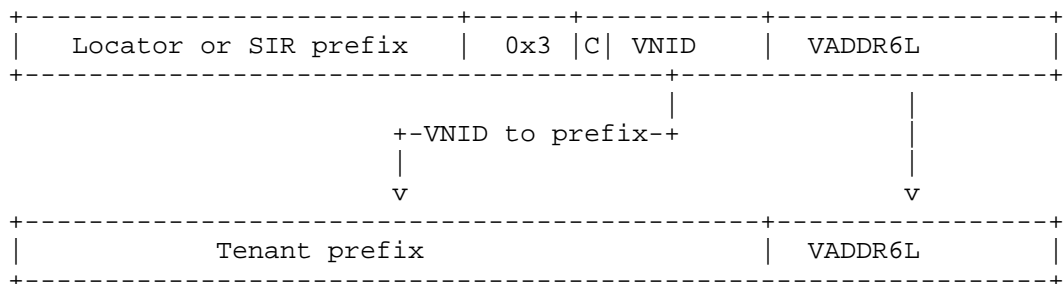


VADDR6L contains the low order 32 bits of the IPv6 virtual address. The upper 96 bits of the virtual address inferred from the VNID to prefix mapping. Note that for ILA translations the low order sixteen of the VADDR6L may be modified for checksum-neutral adjustment.

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

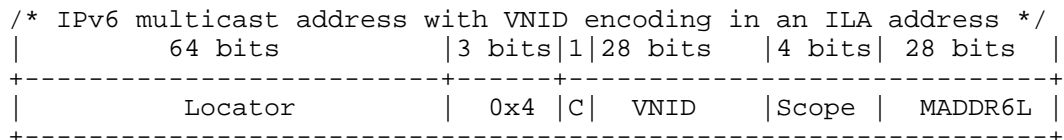


This encoding is reversible, given an ILA address, the virtual address visible to the tenant can be deduced:



4.2.5 Virtual networking identifiers for IPv6 multicast

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



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

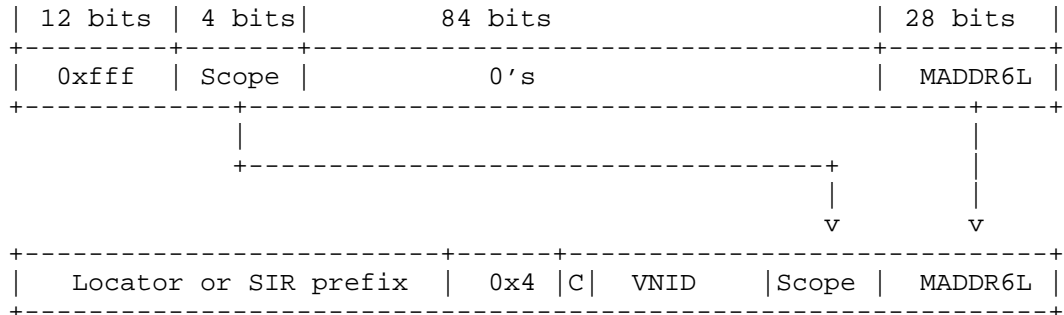
ff0<Scope>::<MADDR6L high 12 bits>:<MADDR6L low 16 bits>

And so can encode multicast addresses of the form:

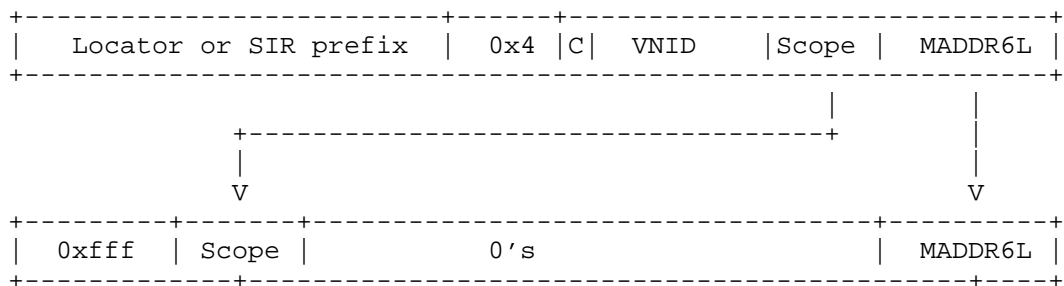
ff0X::0 to ff0X::0fff:ffff

The figure below illustrates encoding a tenant IPv6 virtual multicast

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



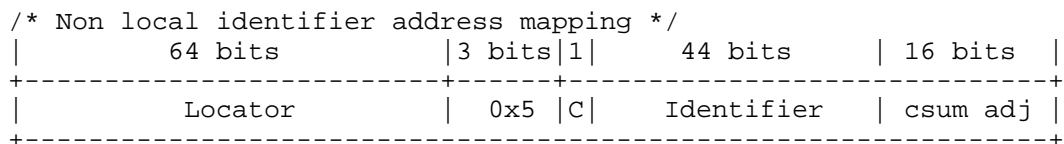
This translation is reversible:



4.2.6 Non-local address identifiers

Non-local address identifiers allow mapping an arbitrary address to an ILA address. The mapping system contains an entry that associates an IPv6 address with an identifier. The associated IP address does not need to be a SIR address or even in the same routing domain.

The format of a non-local address identifier is

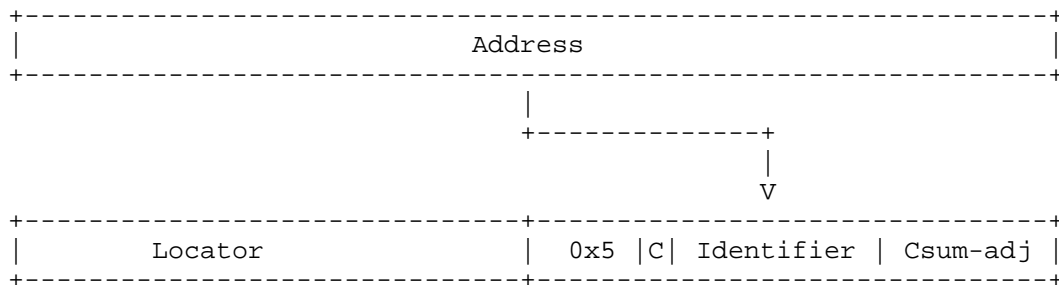


If the checksum adjust field is present it is not part of the identifier that is used in the mapping lookup. The high order bits of the address were originally not a SIR prefix, so it cannot be assumed the checksum adjustment is based on a SIR prefix. The identifier is taken to be the forty-four bits that precede the checksum adjustment

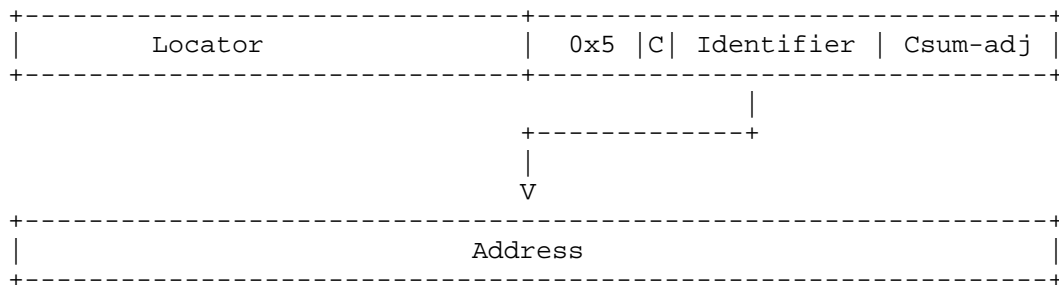
field. When creating the ILA address, the checksum adjustment field is initialized to zero and then set based on checksum difference between the original non-local address and the ILA address.

The figure below illustrates encoding an address into a locator address.

```
/* Non local address identifier */
```

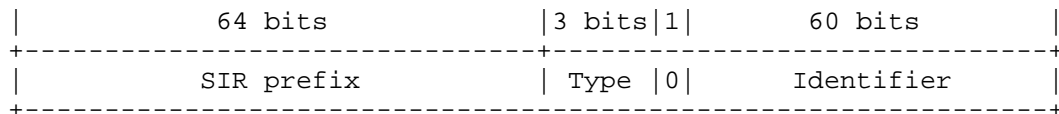


A reverse translation is performed based on a lookup in the mapping table on the identifier (44 bits as shown above). The result of the lookup provides the original address.



4.3 SIR addresses with formatted identifiers

The format of a SIR address with a formatted identifier is:



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

4.3.1 SIR for locally unique identifiers

The SIR address for a locally unique identifier has format:

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

4.3.2 SIR for virtual addresses

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

	64 bits	3 bits 1	28 bits		32 bits	
+-----+		+-----+		+-----+		+-----+
	SIR prefix	0x3 0	VNID		VADDR6	
+-----+		+-----+		+-----+		+-----+

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

4.3.2 SIR for non-local address identifiers

A non-local address identifier can be encoded using the standard identifier representation. For example, an encoding may be:

	64 bits	3 bits 1	44 bits		16 bits	
+-----+		+-----+		+-----+		+-----+
	SIR prefix	0x5 0	Identifier		0	
+-----+		+-----+		+-----+		+-----+

Note that lower order sixteen bits are set to zero since that would be the checksum adjustment value bits if translated to an ILA address.

5 Operation

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

5.1 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 node 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.

5.2 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 section Appendix A for examples).

5.2.1 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 node 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 4.2. 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 5.4).
- 4) Packet is forwarded on the wire. The network routes the packet to the node indicated by the locator.

5.2.2 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 virtual 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 node.
- 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.
- 3) Perform reverse checksum-neutral mapping if C-bit is set (section 5.4).
- 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 the application.

5.3 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.

5.3.1 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.

5.3.2 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.

5.4 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:

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

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

5.4.1 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) If the C-bit is to be used then add-with-carry the bit-wise not of the 0x1000 (i.e. 0xffff) to the value from #1. This compensates the checksum for setting the C-bit.
- 3) Add-with-carry the value from #2 to the low order sixteen bits of the identifier.
- 4) Set the resultant value from #3 in the low order sixteen bits of the identifier and set the C-bit if it is to be present.

Note that the "adjustment" (the 16-bit value set in the identifier) 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 checksum-neutral mapping is applied to the packet (either the C-bit is set or its used is assumed for the ILA domain):

- 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) If the C-bit is used then 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 if its present. This restores the original identifier sent in the packet.

Note that receive checksum-neutral mapping process requires that the original upper sixty four bits in the address can be deduced. The method for this is different based on identifier type:

- o interface identifier: checksum-neutral mapping is not used.
- o locally unique identifier: the SIR prefix is inferred from the one to one mapping with a locator.
- o virtual network identifier for IPv4: the original upper sixty-four bits are assumed to be zero.

- o virtual network identifier for IPv6 unicast: the VNID in the identifier is mapped to a tenant prefix that includes the original upper sixty-four bits.
- o virtual network identifier for IPv6 multicast: the original upper sixty-four bits can be deduced by from the scope field in the identifier and fixed field of the multicast address.
- o non-local address identifier: the identifier, not including the low order sixteen bits of the address, is used to lookup the original address. Since the full address is provided by the lookup, the process to undo a checksum-neutral mapping can be obviated in this case

5.4.2 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.

5.5 Non-local address mapping

Non-local addresses may be mapped into ILA addresses using non-local address identifiers. This allows transit of such addresses across the underlay of an ILA domain. This would be useful for handling addresses in a network that originate from an external source. An example of this would be roaming in cellular network so that a device can continue using addresses that are part of its home network.

A packet may be forwarded to an ILA router that has a non-local destination address which is not a SIR address for the domain. An ILA router can perform a lookup on the full address in an alternate mapping table. If there is a match, an identifier is returned that reverses maps to the address. This identifier is in the ILA domain space and identifies the node with the non-local address. A normal mapping table lookup can then be done to get the locator for the node in the ILA domain.

At a peer ILA router, a lookup is performed on the destination identifier in a table that maps the non-local address identifier to the original non-local address. If an entry is found, the address is set in the destination address and the packet is forward to the destination.

Note that the non-local address to identifier mapping and its reverse mapping must be set in the table before hand.

5.6 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].

5.7 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 with 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 the identifier is unregistered. The control plane should be able to detect a registration attempt for an existing identifier and deny the request.

5.8 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.

5.8.1 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. Match these bits 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 includes checksum-neutral mapping, undo the checksum-neutral mapping using the SIR address found in #1 and the process in section 5.4.1. 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.

5.8.2 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 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 header in 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.

5.9 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.

6 Motivation for ILA

6.1 Use cases

6.1.1 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]).

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.

6.1.2 Datacenter virtualization

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

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

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

6.1.3 Mobile networks

ILA may be applied as a solution for mobility in mobile networks (such as cellular networks). In mobile networks, devices such as smart phones move physically within the network. When a device moves it changes its point of attachment in the network. The goal of mobility is to provide a seamless transition when a device moves from one attachment point to another. Appendix D provides more detail and motivation for ILA in wireless networks.

Each mobile device in a network may be assigned one or more identifiers to use in communications. The ILA mapping table has an entry for each identifier that maps to a locator indicating the current network point of attachment for the device. Nodes that send to the device set the locator per the mapping. When a mobile device moves to a new attachment point, then mapping table entries all of its associated identifiers are updated with a new locator.

6.2 Alternative methods

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

6.2.1 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.

6.2.2 Flow label as virtual network identifier

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

This approach has some issues:

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

6.2.3 Extension headers

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

This technique has some issues:

- o Intermediate devices must not insert extension headers [RFC8200].
- o Extension headers introduce additional packet overhead which may impact performance.
- o Extension headers are not covered by transport checksums (as the

addresses would be) nor any other checksum.

- o Extension headers are not widely supported in network hardware or devices. For instance, several NIC offloads don't work in the presence of extension headers.

6.2.4 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.

7 IANA Considerations

There are no IANA considerations in this specification.

8 References

8.1 Normative References

- [RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <<https://www.rfc-editor.org/info/rfc8200>>.
- [RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006.
- [RFC6296] Wasserman, M. and F. Baker, "IPv6-to-IPv6 Network Prefix Translation", RFC 6296, June 2011.
- [RFC1071] Braden, R., Borman, D., Partridge, C., and W. Plummer, "Computing the Internet checksum", RFC 1071, September 1988.
- [RFC1624] Rijssinghani, A., "Computation of the Internet Checksum via Incremental Update", RFC 1624, May 1994.
- [RFC6724] Thaler, D., Ed., Draves, R., Matsumoto, A., and T. Chown, "Default Address Selection for Internet Protocol Version 6 (IPv6)", RFC 6724, September 2012.

8.2 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.

- [RFC6144] Baker, F., Li, X., Bao, C., and K. Yin, "Framework for IPv4/IPv6 Translation", RFC 6144, April 2011.
- [RFC8014] Black, D., Hudson, J., Kreeger, L., Lasserre, M., and T. Narten, "An Architecture for Data-Center Network Virtualization over Layer 3 (NVO3)", RFC 8014, DOI 10.17487/RFC8014, December 2016, <<https://www.rfc-editor.org/info/rfc8014>>.
- [GUE] Herbert, T., and Yong, L., "Generic UDP Encapsulation", draft-ietf-intarea-gue-04, work in progress.
- [GUESEC] Yong, L., and Herbert, T. "Generic UDP Encapsulation (GUE) for Secure Transport", draft-hy-gue-4-secure-transport-03, work in progress

9 Acknowledgments

The authors would like to thank Mark Smith, Lucy Yong, Erik Kline, Saleem Bhatti, 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; Kalyani Bogineni, Niranjan Avula, and Ratul Guha for insights regarding the mobility use case.

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

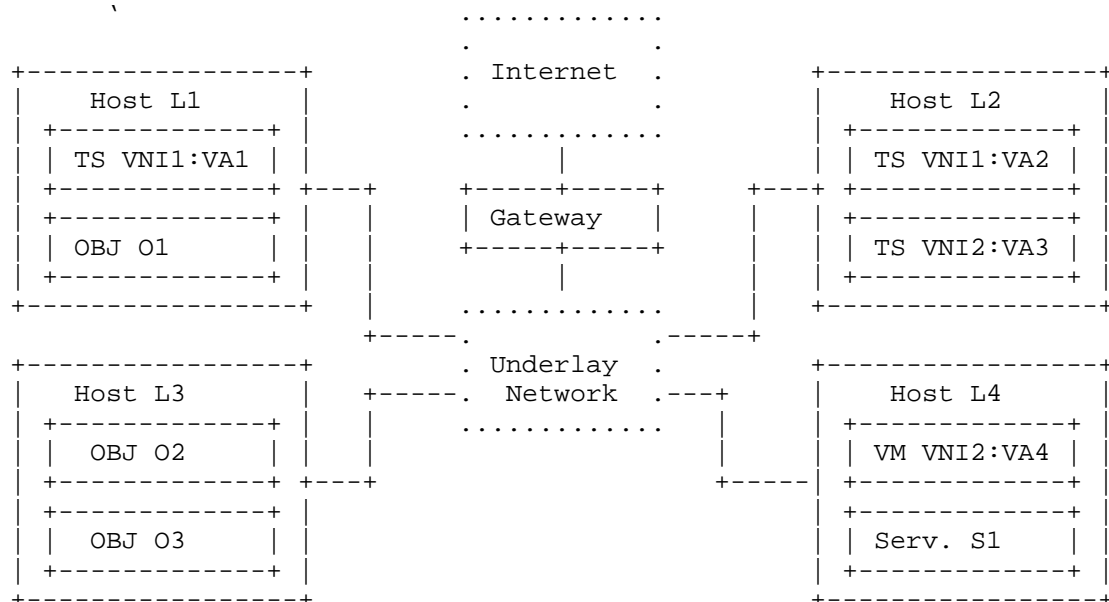
Identifier-locator addressing is broad enough in scope to address many different types of networking entities. For the purposes of this section we classify these as "objects" and "tenant systems".

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
- 14) Non-local address to tenant system

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 task T1 is communicating with task T2, 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
```

A.14 Scenario 14: Non-local address to tenant system

A tenant system may have a global address that is non-local to the network. A host on the Internet or a tenant system may send packet to this address. The packet is forwarded by some means to a gateway or other ILA node (tunneling could be used to accomplish this). An ILA

node can create an ILA address for this using a non-local address identifier.

For a node sending to a non-local address that is an address of task T2, the ILA translation sequence would be:

```
SADDR,A           // Endpoint at a host
SADDR,L3:X        // On the wire
SADDR,A           // Received by TS 2
```

Note that A is the non-local address, and X is an identifier that maps to the non-local address.

Appendix B: unique identifier generation

The unique identifier type of ILA identifiers can address 2^{60} objects (assuming the typed identifier format is used as described in section 4). 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 by the host. The identifier might be composed of a twenty-four bit host identifier followed by a thirty-six bit timestamp. Assuming that a host can allocate up to 100 identifiers per second, this allows about 21.8 years before wrap around.

```

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

```

Appendix C: Datacenter task virtualization

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

C.1 Address per task

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

With ILA, each service task can be assigned its own IPv6 address and therefore will logically be assigned the full port space for that

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

C.2 Job scheduling

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

Appendix D: Mobility in wireless networks

ILA can be used in public wireless networks to provide a solution for mobility.

Devices in a carrier network are referred to as User Equipment (UE) and can include smart phones, automobiles, and other IoT devices. UEs attach to provider network at base stations (eNodeB in carrier terminology). As the device moves, it may change its point of attachment to a geographically close based station. A cellular network is composed of cells each of which has an eNodeB.

A node may change cells several times over a time period. In order to provide seamless communications it is desirable that the existing connections of the device are preserved. ILA provides for this by assigning SIR addresses to UEs and deploying ILA routers in the network infrastructure.

In a canonical architecture each base station (eNodeB) would have an ILA router, and there would be a number of ILA routers that serve as gateways between a provider's network and the Internet. When a host on the Internet sends to a UE's SIR address, a gateway ILA router will translate the address. The locator addresses the base station that is the current point of attachment. At the base station ILA router, the destination is translated back to a SIR address and delivered to a UE. A similar process can happen when two UEs in the network communicate.

The wireless network use case is conceptually similar to network virtualization. In both scenarios, nodes have a point of attachment and can move to other points of attachment. The difference is that in network virtualization it is virtual machines that are mobile, in wireless networks it is real devices.

The wireless use case has some unique properties:

- o These are often public networks so that privacy is a consideration. It is likely that devices may have many addresses assigned to promote privacy.
- o A single device might have many identifiers assigned to it. When a device moves, all of the identifiers must change to map to the same locator.
- o Devices move on their own accord so that mobility is unpredictable.
- o There are mostly real humans using devices so that human

identity and exposing geo location are concerns.

Author's Address

Tom Herbert
Quantonium
4701 Patrick Henry Dr.
Santa Clara, CA
EMail: tom@herbertland.com

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

INTERNET-DRAFT
Intended Status: Informational
Expires: August 2018

T. Herbert
Quantonium

February 20, 2018

Privacy in IPv6 Network Prefix Assignment
draft-herbert-ipv6-prefix-address-privacy-00

Abstract

This document discusses privacy concerns around network prefix assignment in IPv6. It evaluates the privacy threat, proposes a set of ideal criteria for strong privacy, and suggests solutions to achieve a high degree of privacy in addressing.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/lid-abstracts.html>

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

Copyright and License Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1	Introduction	3
2	The privacy concern	3
3	Prior work	4
3.1	SLAAC and DHCPv6-PD	4
3.2	Privacy addresses	4
3.3	Privacy in IPv6 address generation mechanisms	5
3.4	Host address availability recommendations	6
3.5	IPWAVE	6
4	Practical effects	7
4.1	Mobile networks	7
4.2	Connected cars	7
4.3	Privacy implications of NAT	8
4.4	Exploit to defeat prefix rotation	8
5	Criteria for strong privacy	9
6	Identifier/locator split solution	10
6.1	Overview	10
6.2	Scaling identifier/locator address assignment	11
6.2.1	Scaling the amount of mapping state	11
6.2.1.1	Hybrid address assignment	11
6.2.1.2	Hidden aggregation	11
6.2.2	Scaling bulk address assignment	12
6.2.2.1	Bulk assignment using DHCPv6	12
6.2.2.2	Hidden aggregation assignment	13
6.2.3	Practicality of hidden aggregation methods	13
6.3	Law enforcement considerations	14
7	Security considerations	15
8	References	16
8.1	Normative References	16
8.2	Informative References	16
9	Acknowledgments	17
	Authors' Addresses	17

1 Introduction

This document discusses privacy of network prefix assignment in IPv6.

A common address assignment method is for a network to assign prefixes to devices. SLAAC and DHCP-PD are two mechanisms for doing this. In the common case of a /64 assignment (as in SLAAC) the device generates IIDs (interface identifiers) to create individual addresses within an assigned prefix. While significant effort has gone into IID generation techniques to protect privacy ([RFC4941], [RFC7721]), the privacy aspects of the prefix itself have not been fully examined.

This document is focused on privacy within the network layer and specifically with privacy in addressing. There are many other privacy issues that arise from persistent identifiers used in higher (and lower) protocol layers (MAC address, session IDs, certificates, etc.). Discussion of these are out of scope for this document, however it is clear that to achieve a level of privacy that users deserve all layers will need to be considered.

2 The privacy concern

In the original IPv6 addressing model, subnets (links) were assigned a sixty-four bit prefix [RFC4291]. Hosts in the subnet would then generate IIDs that are combined with the subnet prefix to create IPv6 addresses. This model was subsequently extended to assign network prefixes, such as /64s, to general purpose hosts ([RFC3314], [RFC7934]).

When a prefix is assigned to an end host, the prefix becomes an identifier for the host. So, if two such addresses have the same prefix (i.e. same upper sixty-four bits) then they can be assumed to refer to the same host. The IID portion of the addresses (lower sixty-four bits) are immaterial in this inference, so IID generation techniques don't affect the ability to make correlations.

The fact that two addresses can be correlated to be from the same host implies the privacy concern. If an attacker knows that a network provider assigns /64 prefixes to end hosts, as is common in mobile networks, then it can deduce that two addresses in the provider prefix sharing the same sixty-four bit prefix refer to the same host. This correlation can be made between addresses of different flows independently of IIDs in those addresses. Furthermore, with a little more information (see Section 4.3), an attacker may not only deduce two addresses refer to the same end host, but also may be able to discover the identities of individuals in communications.

3 Prior work

Several RFCs describe prefix assignment mechanisms and the privacy and security considerations for them.

3.1 SLAAC and DHCPv6-PD

SLAAC [RFC4862] and DHCPv6-PD [RFC3633] are mechanisms to assign network prefixes to devices. Their respective specifications do not address privacy issues of prefix assignment. Security considerations are focused on the mechanisms.

3.2 Privacy addresses

[RFC4941] addresses issues with persistent identifiers in IPv6. It describes the risks of extended use of the same identifier, and recommends using random interface identifiers and changing addresses periodically to deter inferences to reveal identify, location, or other privacy sensitive attributes of parties in communication. Addresses created by following RFC4941 recommendations are often called "privacy addresses".

RFC4941 is mostly concerned with privacy and security aspects of IID generation. It mentions the problem of privacy of network prefixes in passing:

Although it might appear that changing an address regularly in such environments would be desirable to lessen privacy concerns, it should be noted that the network prefix portion of an address also serves as a constant identifier. All nodes at, say, a home, would have the same network prefix, which identifies the topological location of those nodes. This has implications for privacy, though not at the same granularity as the concern that this document addresses. Specifically, all nodes within a home could be grouped together for the purposes of collecting information. If the network contains a very small number of nodes, say, just one, changing just the interface identifier will not enhance privacy at all, since the prefix serves as a constant identifier.

Nevertheless, it's reasonable that some of the recommendations could be extrapolated to apply to prefix assignment for providing privacy. For instance, RFC4941 suggests to periodically do address rotation by generating a new IID. Conceivably, a node could periodically request a new network prefix via SLAAC. The new prefix would be randomized so that no correlation can be drawn between it and the old prefix.

As for the frequency of changing addresses, RFC4941 states:

having large numbers of clients change their address on a daily or weekly basis is likely to be sufficient to alleviate most privacy concerns.

The statement is neither normative nor quantified. Intuitively, one might assume that a higher frequency of address rotation reduces the probability of privacy being compromised. However, other than the case where a different address is used for each flow (see below), there is no known way to quantify the relationship between frequency of changing addresses and privacy provided to users.

A second concern with recommendations of RFC4941 is that it is was written eleven years ago. The sophistication and capabilities of attackers have increased substantially, so recommendations, such as changing addresses on a daily or weekly basis, may no longer be sufficient even if they were eleven years ago.

Presumably, one could try to achieve a high degree of privacy by changing addresses at a high frequency (every few seconds for instance). The effect on privacy is still unquantifiable, however there is another problem in the disruption caused by changing addresses. An address change would require termination of existing flows, so a high frequency of address rotation would constantly thrash connections. A potential mitigation would be to allow a host to retain network prefixes for which it's still using for flows; however, managing that would be cumbersome and likely wouldn't scale since hosts could accumulate many prefixes over time.

The postulated exploit described in Section 4.4 would defeat the privacy protection of any frequency of address rotation except for the case where a different address is used per flow.

3.3 Privacy in IPv6 address generation mechanisms

[RFC7721] mainly focuses on security and privacy considerations for IID generation. The concern around privacy in network prefix assignment is raised:

As [RFC4941] notes, if a very small number of nodes (say, only one) use a particular prefix for an extended period of time, the prefix itself can be used to correlate the host's activities regardless of how the IID is generated. For example, [RFC3314] recommends that prefixes be uniquely assigned to mobile handsets where IPv6 is used within General Packet Radio Service (GPRS). In cases where this advice is followed and prefixes persist for extended periods of time (or get reassigned to the same handsets

whenever those handsets reconnect to the same network router), hosts' activities could be correlatable for longer periods than the analysis below would suggest.

RFC7721 does not suggest any requirements or guidelines for privacy in network prefixes. Similar to RFC4941, RFC7721 frames the problem with an unquantified description as using a prefix for "extended periods of time".

Note that RFC7721 points out that mobile handsets are often assigned a single prefix. In this case, there is one to one relationship between a prefix and device. For a personal device, such as a smart phone or tablet, there would then be a one to one relationship between a prefix and an individual user.

3.4 Host address availability recommendations

[RFC7934] recommends that general-purpose hosts are assigned multiple globally IPv6 addresses when they attach. RFC7934 advocates prefix assignment and /64 assignment with SLAAC in particular.

RFC7934 includes a section on host tracking (Section 9.1 of RFC7934), however this section focuses on facilitating tracking of hosts in provider networks to satisfy legal requirements.

From RFC7934:

Using SLAAC with a dedicated /64 prefix for each host simplifies tracking, as it does not require logging every address formed by the host

RFC7934 references RFC4941, but does not otherwise address issues with privacy in prefix assignment.

3.5 IPWAVE

[IPWAVE] provides the problem statement for IPWAVE. The issue of address tracking is raised in the Security Considerations section. From the draft:

To prevent an adversary from tracking a vehicle by with its MAC address or IPv6 address, each vehicle should periodically update its MAC address and the corresponding IPv6 address as suggested in [RFC4086][RFC4941]. Such an update of the MAC and IPv6 addresses should not interrupt the communications between a vehicle and an RSU.

As in the RFCs cited above, the draft suggests that addresses should

be changed periodically, however there is no guidance as to what an acceptable frequency of change is to prevent tracking. It is noteworthy that address change is expected to not interrupt communications.

4 Practical effects

This section discusses the current characteristics and effects on privacy in network prefix assignment to hosts.

4.1 Mobile networks

Privacy in prefix addressing is of particular concern in mobile networks. It is often the case that UEs (devices such as smart phones) are assigned a unique /64 prefix that is not shared with other devices. As pointed out by RFC4941 and RFC7721, these network prefixes allow the device to be tracked through correlations. For personal devices, such as smart phones or tablets, correlations on IP addresses could be used to infer user identities in communication. The correlation to a user may require additional information that might be relatively easy to acquire as demonstrated by the exploit described in section 4.4.

Most mobile providers follow the advice of [RFC3314] and assign single a /64 to each device. They may implement a method to force a device to periodically request a new /64 assignment.

A sample implementation in a mobile network could assign a /64 prefix to each IPv6 PDN, and the same prefix is retained for Idle to Active to Idle transitions for the duration of the PDN session. If the UE is idle without transmitting/receiving any packets, the PDN session is dropped when the Idle Timer expires (e.g. 2 hours) and the prefix allocation is released. So in this case the minimum amount of time between addresses change is 2 hrs., but a device could keep its prefix allocation indefinitely as long as the device remains active.

4.2 Connected cars

Connected cars are projected to become ubiquitous over the next decade. By some estimates there will be 381 million connected cars on the road by 2020, and by 2025 all new cars manufactured will be connected. Today many vehicles are already connected to the Internet via 4G LTE, and in the future they will connect using 5G, WiFi, DSRC or other radio technologies. In-vehicle networks connect sensors, displays, navigation, entertainment, as well as personal devices being used by passengers.

Privacy in such a network is potentially a more difficult problem

since there are two independent parties that are involved in address assignment. The vehicle as a mobile node must be assigned addresses by the mobile network, and in turn the vehicle delegates addresses to devices attached to the vehicle network.

A /64 prefix could be assigned to vehicles which is a common mobile network assignment. Devices attached to the vehicle network are delegated IPv6 address within the prefix assigned to the vehicle. This results in all the attached devices sharing fate with respect to privacy. For instance, if an attacker is able to determine the location of just one device with an assigned prefix, then it can infer the location of all devices that share the same prefix. If identity of a user can be separately surmised, this raises the prospect that location of individuals can be tracked.

Periodically changing prefixes in this environment is problematic. As described in Section 3.2, a prefix change is potentially disruptive to communications as this results in an address change for each attached device. In the case of a vehicle network, the attached devices and applications they are running may be very heterogeneous such that their response and recovery for an address change may vary significantly. For instance, a laptop might attach to a vehicle network. A laptop is not normally considered a "mobile device" like a smart phone and many applications they might run don't assume addresses constantly change. Periodically changing addresses for privacy benefit may wreak havoc on such applications.

4.3 Privacy implications of NAT

Network Address Translation (NAT) is a method of remapping one IP address space into another by modifying addresses in the IP header of packets while they are in transit across a routing node. NAT has been extensively deployed to allow hosts that are assigned IPv4 private addresses [RFC1918] to communicate with hosts in the global Internet. NAT has been used to extend the usefulness of IPv4 in the face of address depletion.

A side effect of NAT (possibly accidental) is that NAT modifies addresses such that it obfuscates the identity of the source host behind a NAT. With a significant population of users sharing a pool of NAT addresses, an external observer can draw little correlation based on addresses between flows that have gone through a NAT device. The result is that NAT provides strong privacy in addressing. NAT use is of particular concern to law enforcement since its privacy characteristics complicate criminal investigation [EUROPOL].

4.4 Exploit to defeat prefix rotation

As mentioned in a Section 3.2, one might try to provide privacy in addressing by changing addresses with a high frequency. The following exploit is postulated as a way to defeat the privacy goals of periodic address rotation at any frequency except when a different address is used for each connection.

The exploit is:

- o An attacker creates an "always connected" app that provides some seemingly benign service and users download the app.
- o The app includes some sort of persistent identity. For instance, this could be an account login.
- o The backend server for the app logs the identity and IP address of a user each time they connect.
- o When an address change happens, existing connections on the user device are disconnected. The app will receive a notification and immediately attempt to reconnect using the new source address.
- o The backend server will see the new connection and log the new IP address as being associated to the user. Thus, the server has a real-time record of users and the IP address they are using.
- o The attacker intercepts packets at some point in the Internet. The addresses in the captured packets can be time correlated with the server database to deduce identities of parties in communications that are unrelated to the app.

5 Criteria for strong privacy

A set of "ideal" criteria for strong privacy in addressing can be established. These criteria are intended to be specific, such that when applied to a solution the amount of information that can be inferred by correlating addresses is quantifiable.

The ideal criteria for IPv6 addresses that provide strong privacy are:

- o Addresses are composed of a global routing prefix and a suffix that is internal to an organization or provider. This is the same property for IP addresses [RFC4291].
- o The registry and organization of an address can be determined by the network prefix. This is true for any global address. The organizational bits in the address should have minimal hierarchy to prevent inference. It might be reasonable to have an internal

prefix that divides identifiers based on broad geographic regions, but detailed information such as location, department in an enterprise, or device type should not be encoded in a globally visible address.

- o Given two addresses and no other information, the desired properties of correlating them are:
 - o It can be inferred if they belong to the same organization and registry. This is true for any two global IP addresses.
 - o It may be inferred that they belong to the same broad grouping, such as a geographic region, if the information is encoded in the organizational bits of the address.
 - o No other correlation can be established. It cannot be inferred that the IP addresses address the same node, the addressed nodes reside in the same subnet, rack, or department, or that the nodes for the two addresses have any geographic proximity to one another.

Note that if NAT is deployed with a sufficiently large population of users sharing a pool of IP addresses then these criteria are met. Thus NAT can be considered a baseline for strong privacy in addressing.

6 Identifier/locator split solution

This section proposes using identifier/locator split to meet the strong privacy criteria for addressing in IPv6.

6.1 Overview

Identifier/locator split separates the notions of location and identity in IP addresses. Identifier addresses are addresses that don't contain topological information for routing within a network. Nodes are assigned identifier addresses that can be used as endpoints in communications. Locator addresses indicate the topological location of a logical node. In order to forward a packet to a destination with an identifier address, an ingress node for a network maps an identifier address to a locator address. A network overlay method is used to forward the packet to the location in the network of the logical or mobile node.

Since identifier addresses are non-topological they don't require any hierarchy in address assignment beyond the global network prefix. Therefore the network can randomly generate identifier addresses within a portion of the address in a space of at least sixty-four

bits.

Strong privacy in addressing can be achieved by using a different randomly generated identifier address for each flow. Conceptually, this would entail that the network creates and assigns a unique and untrackable address to a host for every flow created by a host. Some suggestions for scaling this technique are discussed below.

Note that this technique parallels what NAT does in that NAT effectively creates a different source address per connection. Unlike NAT however, address assignments in identifier/locator split are stateless in the network and transparent to the end points.

6.2 Scaling identifier/locator address assignment

Assigning an address per connection is a potential scaling problem on two accounts:

- o The amount of state needed in the mapping system is significant.
- o Bulk host address assignment is inefficient.

6.2.1 Scaling the amount of mapping state

The amount of state necessary to assign each flow its own unique source IP address is equivalent, or at least proportional, to the amount of state needed for NAT-- basically this is one state element for every connection in the network. So in one sense this solution should scale as well as NAT has.

6.2.1.1 Hybrid address assignment

Not all communications might require strong privacy, so it is conceivable that a hybrid approach to address assignment might be taken. A network might assign prefixes for use with communications that are not privacy sensitive, and may assign singleton addresses that meet strong privacy criteria for privacy sensitive communications. Assuming that most communications don't need strong privacy this could reduce the amount of state needed in the mapping system considerably. The decision as to whether strong privacy is required for a communication would be made by the user or application.

6.2.1.2 Hidden aggregation

A possible solution to reduce state is to make addresses aggregable, but use an aggregation method that is known only by the network provider and hidden to the rest of the world. The network could use a

reversible hash or encryption function to create addresses.

The input to an address generation function includes a device identifier, a secret key, and a generation index.

The function may have the form:

$$\text{Address} = \text{Func}(\text{key}, \text{dev_ident}, \text{gen})$$

Where "key" is secret to network, "dev_ident" is a network internal identifier for a device (roughly equivalent to "identity" in IDEAS), and "gen" is generation number 0,1,2,... N. The generation value is changed for each invocation to create different addresses for assignment to a device.

When a network ingress node is forwarded a packet it performs the inverse function on an address.

The inverse function has the form:

$$(\text{dev_ident}, \text{gen}) = \text{FuncInv}(\text{key}, \text{Address})$$

The returned dev_ident value is used as the identifier in the mapping lookup for a locator address. In this manner, the network can generate many addresses to assign to a device where they all share a single entry in the mapping system.

6.2.2 Scaling bulk address assignment

Assigning multiple addresses without aggregation is difficult to scale. Each address would need to be individually specified in an assignment sent to a host.

6.2.2.1 Bulk assignment using DHCPv6

DHCPv6 might allow bulk singleton address assignment. As stated in [RFC7934]:

Most DHCPv6 clients only ask for one non-temporary address, but the protocol allows requesting multiple temporary and even multiple non-temporary addresses, and the server could choose to provide multiple addresses. It is also technically possible for a client to request additional addresses using a different DHCP Unique Identifier (DUID), though the DHCPv6 specification implies that this is not expected behavior ([RFC3315], Section 9). The DHCPv6 server will decide whether to grant or reject the request based on information about the client, including its DUID, MAC address, and more. The maximum number of IPv6

addresses that can be provided in a single DHCPv6 packet, given a typical MTU of 1500 bytes or smaller, is approximately 30.

6.2.2.2 Hidden aggregation assignment

By extending the concept of hidden aggregation assignment (section 6.2.1.2), it is conceptually possible that a host could work in concert with the network to generate addresses that meet strong privacy criteria. In this method, a host autonomously generates addresses as needed. The network, but no one outside the network, is then able to aggregate the addresses as belonging to the device.

End hosts are generally considered untrusted nodes by the network, so they cannot be given access to the network secret key used for the address generation function. Public key encryption might be used.

A host may perform an encryption function to generate addresses:

```
Address = Encrypt(pub_key, dev_inet, gen)
```

Where "pub_key" is a public key for the network, "dev_inet" is a network identifier for the device and is visible to the device (so it may be leaked). "gen" is a generation number 0,1,2,... N. The generation value is changed for each invocation to create different addresses.

When a network ingress node is forwarded a packet it decrypts an address using the network private key.

The decryption function has the form:

```
(dev_inet, gen) = decrypt(priv_key, Address)
```

Where "priv_key" is the secret private key of the network associated with the public key. The returned dev_inet value is used as the identifier in the mapping lookup for a locator address.

Note that this method would require a new address assignment protocol.

6.2.3 Practicality of hidden aggregation methods

The premise of hidden aggregation is that only trusted devices in the network are able to decode the aggregation hidden within IPv6 addresses. This implies that the network must keep secrets about the process. In the above examples, the secrets are keys used in the hash or encryption. The security of the key is then paramount, so techniques for key management, rotation, and using different key sets for

obfuscation are pertinent.

To perform a mapping lookup a node must apply the inverse address generation function to map addresses to locators. This lookup would occur in the critical data path so performance is important. Encryption and hashing are notoriously time consuming and computationally complex functions.

Some possible mitigating factors for performance impact are:

- o The input to address generation functions is a small amount of data and has fixed size. The input is a key (presumably 128 or 256 bits), part of all of an IPv6 address (128 bits), and a generation number (sixteen to twenty-four bits should work).
- o Given that the input is fixed size, specialized hardware might be used to optimize performance of the inverse address generation function. For instance, modern CPUs include instructions to perform crypto [AES-NI]. Since the keys used in these functions are secret to the network and there are relatively few of them, they might be preloaded into a crypto engine to reduce setup costs.
- o The output of an inverse address generation function is cacheable. A cache on a device could contain address to locator mappings. When the inverse function and lookup on dev_ident are performed, a mapping of address to the discovered locator could be created in the cache. The device could then map addresses in subsequent packets sent on the same flow to the proper locator by looking up the address in the cache.

6.3 Law enforcement considerations

This section discusses law enforcement considerations for host tracking when using an identifier/locator split solution for strong privacy. NAT is used as a reference point for discussion.

There are two sub-problems expressed by law enforcement about NAT [EUROPOL]:

- 1) It is difficult to map a NAT address and port back to a user.
- 2) Many Internet servers do not log the client source port of connections.

The first problem is one of maintaining a log of NAT mappings. If the log contains the inner address, outer address and port, and timestamp when the NAT mapping was created-- then given the log and a NATed

packet, the original sender can be revealed. Note that NAT logs are kept internal to the provider network, and securing them is the responsibility of the provider. The same model can be applied to identifier/locator split where the infrastructure keeps a log of identifier to locator mappings and a timestamp for when they were created.

In the second problem, the source port is needed to be logged in servers in order correlate a flow to an entry in the NAT logs of a provider. The source port is relevant to a NAT mapping; however, in identifier/locator split it's not since identification of a host node contained with an address. Therefore the client source port is not required for tracking users in an identifier/locator solution.

7 Security considerations

The subject of this draft is privacy assigning network prefixes. Implicit to this is that any address assignment technique requires security on the parties entities involved.

In the identifier/locator split the mapping of identifier to locator is privacy sensitive information. The locator may very well imply the geo location of a device. As such, it is recommended that locators that might contain accurate location information are strictly contained within a trusted infrastructure.

In mobile environments, it is natural to group identifiers (addresses) together that have the same attributes [IDGROUP]. For instance, if as in section 6.1 a different source address is used for each flow, all of the addresses assigned to a device form a group. When the device moves, all of the addresses move with it; this can be efficiently implemented as single operation on the mapping system. The group information is thus privacy sensitive information that must be secured by the infrastructure to prevent use of the information to make inferences of identity similar to /64 assignment.

Hidden aggregation is a means of grouping identifiers together similar to the above description. The secret keys used in these algorithms are thus critical information that must be kept secure. Security by obscurity should be avoided here, divulging the algorithm used to generate addresses should not reduce security or privacy.

End hosts must implement appropriate security to ensure privacy. For instance, if an address is assigned per flow as described in Section 6.2, applications must be isolated from one another so that they cannot infer addresses or privacy properties of other applications running within the same system. Also, if a host is completely compromised then that fact should not impact the privacy and security

of other hosts and applications within a network.

8 References

8.1 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, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <<https://www.rfc-editor.org/info/rfc8200>>.

8.2 Informative References

- [RFC4941] Narten, T., Draves, R., and S. Krishnan, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 4941, DOI 10.17487/RFC4941, September 2007, <<https://www.rfc-editor.org/info/rfc4941>>.
- [RFC7721] Cooper, A., Gont, F., and D. Thaler, "Security and Privacy Considerations for IPv6 Address Generation Mechanisms", RFC 7721, DOI 10.17487/RFC7721, March 2016, <<https://www.rfc-editor.org/info/rfc7721>>.
- [RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <<https://www.rfc-editor.org/info/rfc4291>>.
- [RFC3314] Wasserman, M., Ed., "Recommendations for IPv6 in Third Generation Partnership Project (3GPP) Standards", RFC 3314, DOI 10.17487/RFC3314, September 2002, <<https://www.rfc-editor.org/info/rfc3314>>.
- [RFC7934] Colitti, L., Cerf, V., Cheshire, S., and D. Schinazi, "Host Address Availability Recommendations", BCP 204, RFC 7934, DOI 10.17487/RFC7934, July 2016, <<https://www.rfc-editor.org/info/rfc7934>>.
- [RFC4862] Thomson, S., Narten, T., and T. Jinmei, "IPv6 Stateless Address Autoconfiguration", RFC 4862, DOI 10.17487/RFC4862, September 2007, <<https://www.rfc-editor.org/info/rfc4862>>.

- [RFC3633] Troan, O. and R. Droms, "IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) version 6", RFC 3633, DOI 10.17487/RFC3633, December 2003, <<https://www.rfc-editor.org/info/rfc3633>>.
- [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, <<https://www.rfc-editor.org/info/rfc3315>>.
- [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, <<https://www.rfc-editor.org/info/rfc1918>>.
- [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, <<https://www.rfc-editor.org/info/rfc3315>>.
- [EUROPOL] EUROPOL/EC3 to delegations of the Council of the European Union, "Carrier-Grade Network Address Translation (CGN) and the Going Dark Problem", January 2017
- [AES-NI] Gueron, S., "Intel Advanced Encryption Standard (AES) New Instructions", <<https://software.intel.com/sites/default/files/article/165683/aes-wp-2012-09-22-v01.pdf>>
- [IDGROUP] Herbert, T., "Identifier groups", draft-herbert-idgroups-00

9 Acknowledgments

The author would like to thank Robert Moskowitz for insightful comments and contributions to this draft.

Authors' Addresses

Tom Herbert
Quantonium
Santa Clara, CA
USA

Email: tom@quantonium.net

Internet Area WG
Internet-Draft
Intended status: Standard track
Expires June 30, 2017

T. Herbert
Quantonium
L. Yong
Huawei USA
O. Zia
Microsoft
December 30, 2017

Generic UDP Encapsulation
draft-ietf-intarea-gue-05

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>

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

This Internet-Draft will expire on June 30, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

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

Abstract

This specification describes Generic UDP Encapsulation (GUE), which is a scheme for using UDP to encapsulate packets of different IP protocols for transport across layer 3 networks. By encapsulating packets in UDP, specialized capabilities in networking hardware for efficient handling of UDP packets can be leveraged. GUE specifies basic encapsulation methods upon which higher level constructs, such 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	5
1.1. Terminology and acronyms	5
1.2. Requirements Language	6
2. Base packet format	7
2.1. GUE variant	7
3. Variant 0	7
3.1. Header format	8
3.2. Proto/ctype field	9
3.2.1 Proto field	9
3.2.2 Ctype field	10
3.3. Flags and extension fields	10
3.3.1. Requirements	10
3.3.2. Example GUE header with extension fields	11
3.4. Private data	12
3.5. Message types	12
3.5.1. Control messages	12
3.5.2. Data messages	13
3.6. Hiding the transport layer protocol number	13
4. Variant 1	14
4.1. Direct encapsulation of IPv4	14
4.2. Direct encapsulation of IPv6	15
5. Operation	15
5.1. Network tunnel encapsulation	16
5.2. Transport layer encapsulation	16
5.3. Encapsulator operation	16
5.4. Decapsulator operation	17
5.4.1. Processing a received data message	17
5.4.2. Processing a received control message	18
5.5. Router and switch operation	18
5.6. Middlebox interactions	18
5.6.1. Inferring connection semantics	19
5.6.2. NAT	19
5.7. Checksum Handling	19

5.7.1. Requirements	19
5.7.2. UDP Checksum with IPv4	20
5.7.3. UDP Checksum with IPv6	20
5.8. MTU and fragmentation	21
5.9. Congestion control	21
5.10. Multicast	22
5.11. Flow entropy for ECMP	22
5.11.1. Flow classification	22
5.11.2. Flow entropy properties	23
5.12 Negotiation of acceptable flags and extension fields	24
6. Motivation for GUE	24
6.1. Benefits of GUE	24
6.2 Comparison of GUE to other encapsulations	25
7. Security Considerations	26
8. IANA Considerations	27
8.1. UDP source port	27
8.2. GUE variant number	28
8.3. Control types	28
8.4. Flag-fields	28
9. Acknowledgements	29
10. References	29
10.1. Normative References	29
10.2. Informative References	30
Appendix A: NIC processing for GUE	32
A.1. Receive multi-queue	32
A.2. Checksum offload	33
A.2.1. Transmit checksum offload	33
A.2.2. Receive checksum offload	34
A.3. Transmit Segmentation Offload	34
A.4. Large Receive Offload	35
Appendix B: Implementation considerations	36
B.1. Privileged ports	36
B.2. Setting flow entropy as a route selector	36
B.3. Hardware protocol implementation considerations	36
Authors' Addresses	37

1. Introduction

This specification describes Generic UDP Encapsulation (GUE) which is a general method for encapsulating packets of arbitrary IP protocols within User Datagram Protocol (UDP) [RFC0768] packets. Encapsulating packets in UDP facilitates efficient transport across networks. Networking devices widely provide protocol specific processing and optimizations for UDP (as well as TCP) packets. Packets for atypical IP protocols (those not usually parsed by networking hardware) can be encapsulated in UDP packets to maximize deliverability and to leverage flow specific mechanisms for routing and packet steering.

GUE provides an extensible header format for including optional data in the encapsulation header. This data potentially covers items such as 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. [GUEEXTEN] specifies a set of core extensions.

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
GUE variant	A version of the GUE protocol or an alternate form of a version
Encapsulator	A network node that encapsulates packets 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

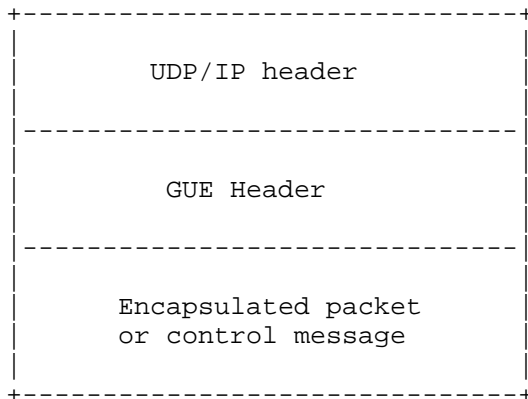
Control message	A formatted message in the GUE payload that is implicitly addressed to the decapsulator to monitor or control the state or behavior of a tunnel
Flags	A set of bit flags in the primary GUE header
Extension field	An optional field in a GUE header whose presence is indicated by corresponding 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 variant

The first two bits of the GUE header contain the GUE protocol variant number. The variant number can indicate the version of the GUE protocol as well as alternate forms of a version.

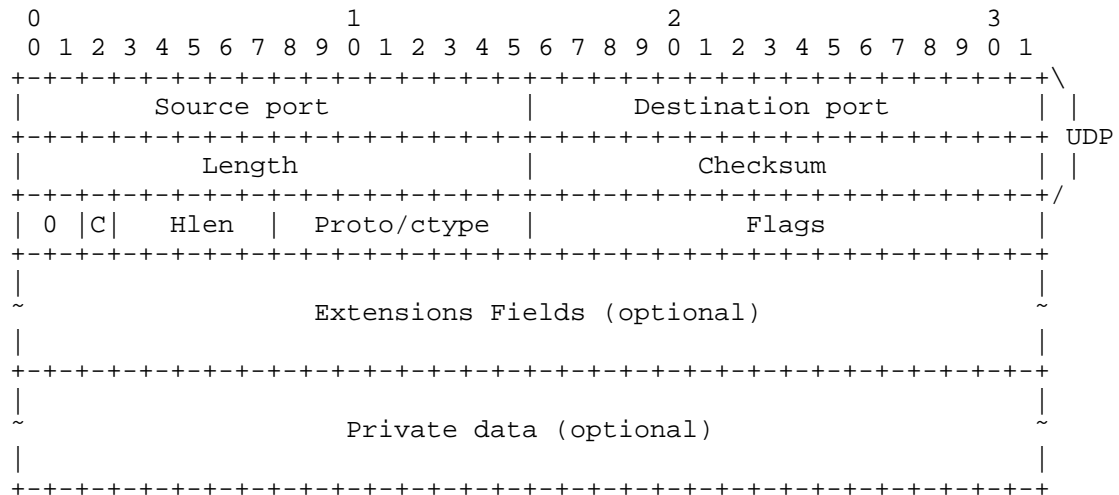
Variants 0 and 1 are described in this specification; variants 2 and 3 are reserved.

3. Variant 0

Variant 0 indicates version 0 of GUE. This variant defines a generic extensible format to encapsulate packets by Internet protocol number.

3.1. Header format

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



The contents of the UDP header are:

- o Source port: If connection semantics (section 5.6.1) are applied to an encapsulation, this is set to the 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 Multit-Path [RFC2992]); the properties of flow entropy are described in section 5.11.
- o Destination port: If connection semantics (section 5.6.1) are applied to an encapsulation, this is set to the destination port for the tuple. If connection semantics are not applied this is set to the GUE assigned port number, 6080.
- o Length: Canonical length of the UDP packet (length of UDP header and payload).
- o Checksum: Standard UDP checksum (handling is described in section 5.7).

The GUE header consists of:

- o Variant: 0 indicates GUE protocol version 0 with a header.
- o C: C-bit: When set indicates a control message, not set indicates a data message.

- o Hlen: Length in 32-bit words of the GUE header, including optional extension fields but not the first four bytes of the header. Computed as $(\text{header_len} - 4) / 4$, 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.
- o Proto/ctype: When the C-bit is set, this field contains a control message type for the payload (section 3.2.2). When the C-bit is not set, the field holds the Internet protocol number for the encapsulated packet in the payload (section 3.2.1). The control message or encapsulated packet begins at the offset provided by Hlen.
- o Flags: Header flags that may be allocated for various purposes and may indicate presence of extension fields. Undefined header flag bits MUST be set to zero on transmission.
- o Extension Fields: Optional fields whose presence is indicated by corresponding flags.
- o Private data: Optional private data 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 standards. Types 128 through 255 are reserved to be user defined for experimentation or private control messages.

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

3.3. Flags and extension fields

Flags and associated extension fields are the primary mechanism of extensibility in GUE. As mentioned in section 3.1, GUE header flags 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. Flags may indicate 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.

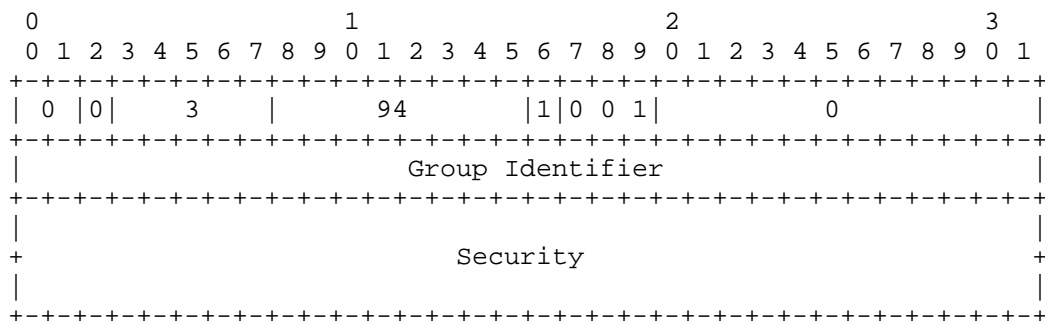
Extension fields are placed in order of the flags. New flags are to be allocated from high to low order bit contiguously without holes. Flags allow random access, for instance to inspect the field corresponding to the Nth flag bit, an implementation only considers the previous N-1 flags to determine 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 [GUEEXTEN] 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 a new 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 Group Identifier 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 Group Identifier extension is present which 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 seven 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:

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

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

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

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

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

3.6. Hiding the transport layer protocol number

The GUE header indicates the Internet protocol of the encapsulated packet. 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, [GUEEXTEN]). 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:

```

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

```

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.

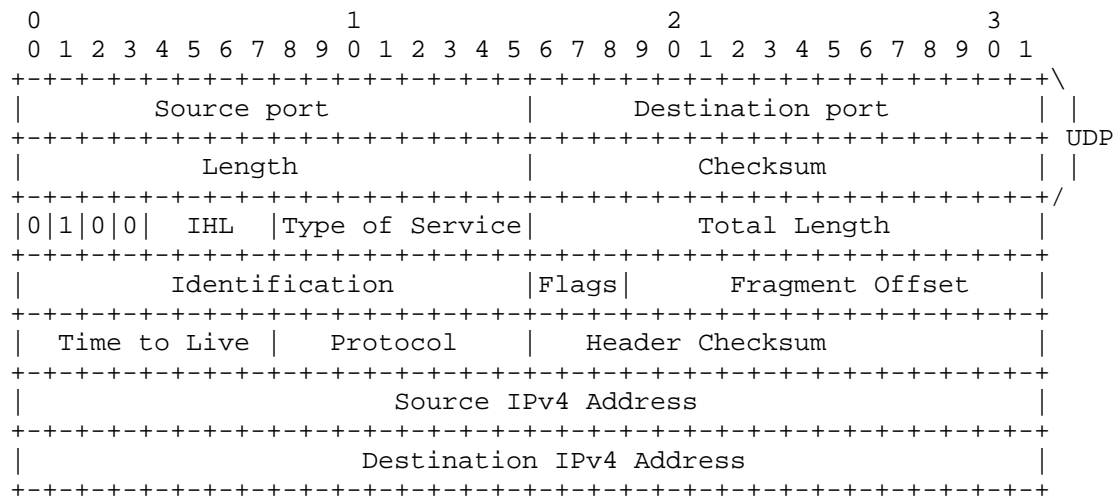
4. Variant 1

Variant 1 of GUE allows direct encapsulation of IPv4 and IPv6 in UDP. In this variant 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 variant 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 variant 1 for direct IP encapsulation which makes two bits of GUE variant to also be 01.

This technique is effectively a means to compress out the version 0 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 variant 0 packets). This technique saves encapsulation overhead on costly links for the common use of IP encapsulation, and also obviates the need to allocate a separate port number for IP-over-UDP encapsulation.

4.1. Direct encapsulation of IPv4

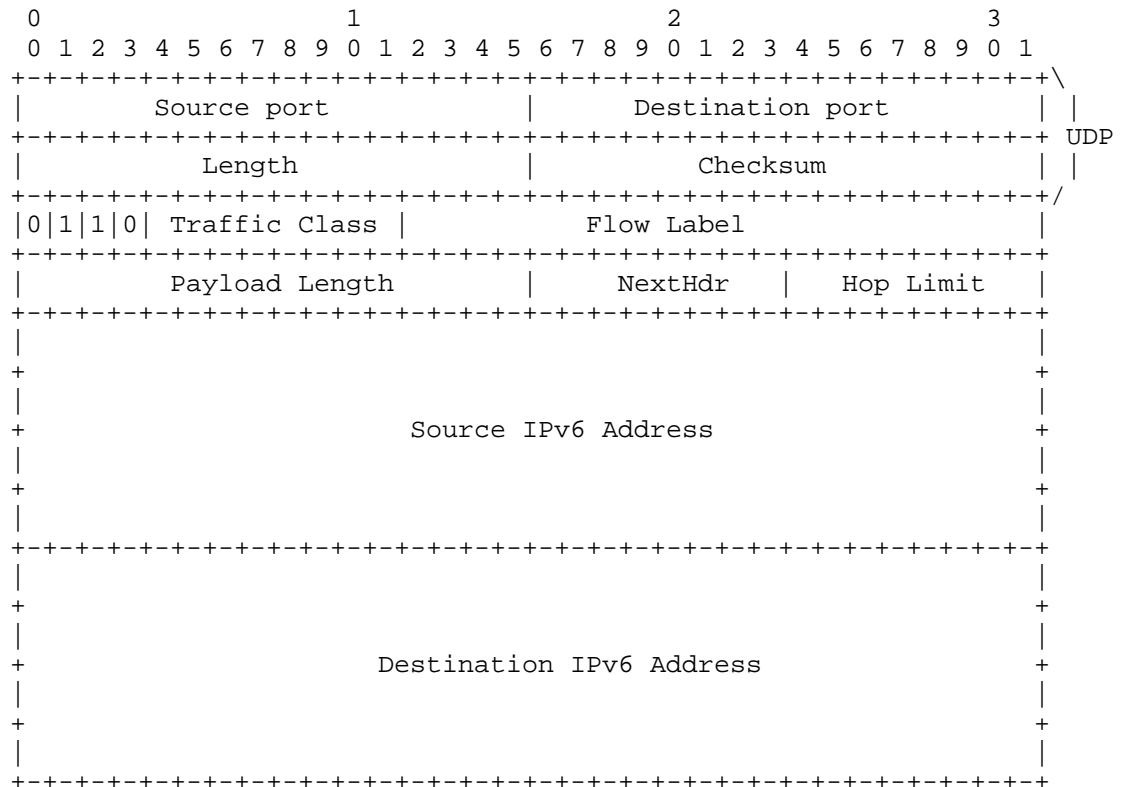
The format for encapsulating IPv4 directly in UDP is:



Note that the 0100 value in the first four bits of the the UDP payload expresses the GUE variant as 1 (bits 01) and IP version as 4 (bits 0100).

4.2. Direct encapsulation of IPv6

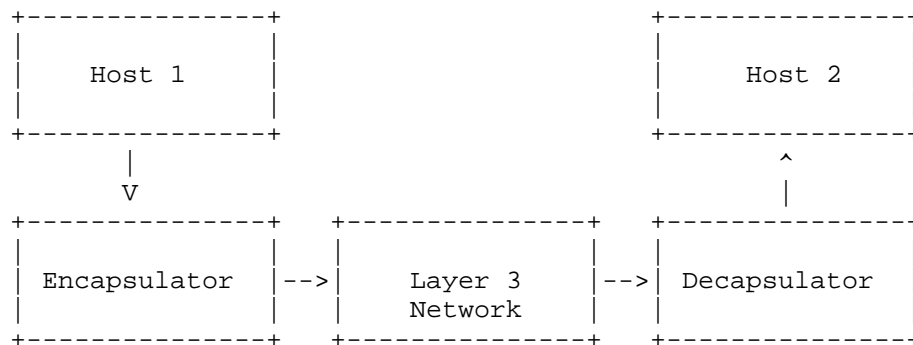
The format for encapsulating IPv6 directly in UDP is demonstrated below:



Note that the 0110 value in the first four bits of the the UDP payload expresses the GUE variant 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, ESP tunnel mode) -- it SHOULD follow standard conventions for tunneling of one protocol over another. For instance, if an IP packet is being encapsulated in GUE then diffserv interaction [RFC2983] and ECN propagation for tunnels [RFC6040] SHOULD be followed.

5.4. Decapsulator operation

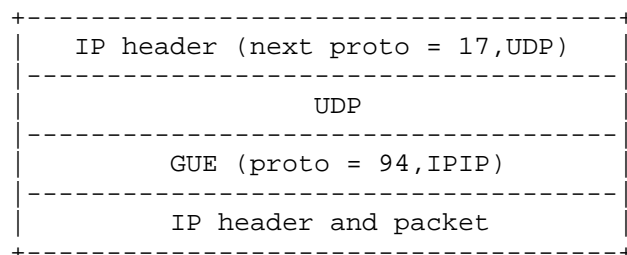
A decapsulator performs decapsulation of GUE packets. A decapsulator is addressed by the outer destination IP address of a GUE packet. The decapsulator validates packets, including fields of the GUE header.

If a decapsulator receives a GUE packet with an unsupported variant, 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.

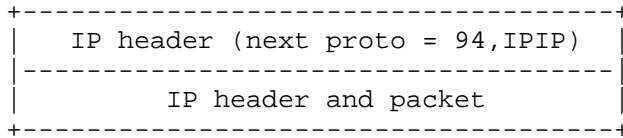
5.4.1. Processing a received data message

If a valid data message is received, the UDP header and GUE header are 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 value in the ctype field of 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 router 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. In cases where the outer five-tuple does not provide sufficient entropy for flow classification, for instance UDP ports are fixed to provide connection semantics (section 5.6.1), then the encapsulated packet MAY be parsed to determine flow entropy.

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 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. The default **MUST** be to set the flow entropy value in the UDP source port.

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:

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

- o The GUE header checksum is used (defined in [GUEEXTEN]).
- o 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 [GUEEXTEN] assuming there are no other mechanisms used to protect the GUE packet.

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 traversing 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 unless there are alternative mechanisms in use that protect against misdelivery. 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.

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

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 [RFC8084], 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:

- o If the encapsulated packet is a layer 4 packet, TCP/IPv4 for instance, the flow entropy could be based on the canonical five-tuple hash of the inner packet.
- o If the encapsulated packet is an AH transport mode packet with TCP as next header, the flow entropy could be a hash over a three-tuple: TCP protocol and TCP ports of the encapsulated packet.
- o If a node is encrypting a packet using ESP tunnel mode and GUE encapsulation, the flow entropy could be based on the contents of 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:

- o 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.
- o The flow entropy has a uniform distribution across encapsulated flows.
- o An encapsulator MAY occasionally change the flow entropy used for an inner flow per its discretion (for security, route selection, etc). To avoid thrashing or flapping the value, the flow entropy used for a flow SHOULD NOT change more than once every thirty seconds (or a configurable value).
- o Decapsulators, or any networking devices, SHOULD NOT attempt to interpret flow entropy as anything more than an opaque value. Neither should they attempt to reproduce the hash calculation used by an encapsulator in creating a flow entropy value. They MAY use the value to match further receive packets for steering decisions, but MUST NOT assume that the hash uniquely or permanently identifies a flow.

- o Input to the flow entropy calculation is not restricted to ports and addresses; input could include flow label from an IPv6 packet, SPI from an ESP packet, or other flow related state in the encapsulator that is not necessarily conveyed in the packet.
- o The assignment function for flow entropy SHOULD be randomly seeded to mitigate denial of service attacks. The seed SHOULD be changed periodically.

5.12 Negotiation of acceptable flags and extension fields

An encapsulator and decapsulator need to achieve agreement about GUE parameters that will be used in communications. Parameters include supported GUE variants, 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.

General methods for this are:

- o Configuration. The parameters used for a tunnel are configured at each endpoint.
- o Negotiation. A tunnel negotiation can be performed. This could be accomplished in-band of GUE using control messages or private data.
- o Via a control plane. Parameters for communicating with a tunnel endpoint can be set in a control plane protocol (such as that needed for network virtualization).
- 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. Standardized

optional data such as security, virtual networking identifiers, fragmentation are being defined.

- * For extensibility, GUE uses flag fields as opposed to TLVs as some other encapsulation protocols do. Flag fields are strictly ordered, allow random access, and are efficient in 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], GENEVE [GENEVE], 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:

- o UDP encapsulation leverages specialized network device processing for efficient transport. The semantics for using the UDP source port for flow entropy as input to ECMP are defined in section 5.11.
- o GUE permits encapsulation of arbitrary IP protocols, which includes layer 2, 3, and 4 protocols.

- o Multiple protocols can be multiplexed over a single UDP port number. This is in contrast to techniques to encapsulate protocols over UDP using a protocol specific port number (such as ESP/UDP, GRE/UDP, SCTP/UDP). GUE provides a uniform and extensible mechanism for encapsulating 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 instance, a TCAM can be used to parse a known set of N flags where the number of entries in the TCAM is 2^N . By comparison, the number of TCAM entries needed to parse a set of N arbitrarily ordered TLVS is 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 [GUEEXTEN]. 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. This extension can be used to perform DTLS 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 <tom@herbertland.com>
Contact: Tom Herbert <tom@herbertland.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 variant number

IANA is requested to set up a registry for the GUE variant number. The GUE variant 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].

Variant number	Description	Reference
0	GUE Version 0 with header	This document
1	GUE Version 0 with direct IP encapsulation	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]. New flags should be allocated from high to low order bit contiguously without holes. [GUEXTENS] requests an initial set of flag assignments.

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>>.
- [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <<http://www.rfc-editor.org/info/rfc1122>>.
- [RFC2434] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 2434, DOI 10.17487/RFC2434, October 1998, <<http://www.rfc-editor.org/info/rfc2434>>.
- [RFC2983] Black, D., "Differentiated Services and Tunnels", RFC 2983, DOI 10.17487/RFC2983, October 2000, <<http://www.rfc-editor.org/info/rfc2983>>.
- [RFC6040] Briscoe, B., "Tunnelling of Explicit Congestion Notification", RFC 6040, DOI 10.17487/RFC6040, November 2010, <<http://www.rfc-editor.org/info/rfc6040>>.
- [RFC6935] Eubanks, M., Chimento, P., and M. Westerlund, "IPv6 and UDP Checksums for Tunneled Packets", RFC 6935, DOI 10.17487/RFC6935, April 2013, <<http://www.rfc-editor.org/info/rfc6935>>.
- [RFC6936] Fairhurst, G. and M. Westerlund, "Applicability Statement for the Use of IPv6 UDP Datagrams with Zero Checksums", RFC 6936, DOI 10.17487/RFC6936, April 2013, <<http://www.rfc-editor.org/info/rfc6936>>.
- [RFC4459] Savola, P., "MTU and Fragmentation Issues with In-the-Network Tunneling", RFC 4459, DOI 10.17487/RFC4459, April 2006, <<http://www.rfc-editor.org/info/rfc4459>>.

10.2. Informative References

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

(ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, DOI 10.17487/RFC5245, April 2010, <<http://www.rfc-editor.org/info/rfc5245>>.

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

- [RFC8084] Fairhurst, G., "Network Transport Circuit Breakers", BCP 208, RFC 8084, DOI 10.17487/RFC8084, March 2017, <<https://www.rfc-editor.org/info/rfc8084>>.
- [GUEEXTEN] Herbert, T., Yong, L., and Templin, F., "Extensions for Generic UDP Encapsulation" draft-herbert-gue-extensions-00
- [GUE4NVO3] Yong, L., Herbert, T., Zia, O., "Generic UDP Encapsulation (GUE) for Network Virtualization Overlay" draft-hy-nvo3-gue-4-nvo-03
- [GUESEC] Yong, L., Herbert, T., "Generic UDP Encapsulation (GUE) for Secure Transport" draft-hy-gue-4-secure-transport-03
- [TCPUDP] Chesire, S., Graessley, J., and McGuire, R., "Encapsulation of TCP and other Transport Protocols over UDP" draft-cheshire-tcp-over-udp-00
- [TOU] Herbert, T., "Transport layer protocols over UDP" draft-herbert-transports-over-udp-00
- [GENEVE] Gross, J., Ed., Ganga, I. Ed., and Sridhar, T., "Geneve: Generic Network Virtualization Encapsulation", draft-ietf-nvo3-geneve-05
- [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 [GUEEXTEN]). Remote checksum offload relies on NIC offload of the

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

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

A.2.2. Receive checksum offload

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

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

A.3. Transmit Segmentation Offload

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

The process of TSO can be generalized as:

- Split the TCP payload into segments which allow packets with size less than or equal to MTU.
- For each created segment:
 1. Replicate the TCP header and all preceding headers of the original packet.
 2. Set payload length fields in any headers to reflect the length of the segment.
 3. Set TCP sequence number to correctly reflect the offset of the TCP data in the stream.
 4. Recompute and set any checksums that either cover the payload of the packet or cover header which was changed by setting a payload length.

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

To facilitate TSO with GUE, it is recommended that extension fields 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

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

B.1. Privileged ports

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

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

B.2. Setting flow entropy as a route selector

An encapsulator generating flow entropy in the UDP source port 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 as 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 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-DRAFT
Intended Status: Proposed Standard
Expires: July 22, 2018

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

January 18, 2018

Extensions for Generic UDP Encapsulation
draft-ietf-intarea-gue-extensions-03

Abstract

This specification defines a set of the fundamental optional extensions for Generic UDP Encapsulation (GUE).

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/lid-abstracts.html>

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

Copyright and License Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction	4
2. GUE header format with optional extensions	4
3. Group identifier option	6
3.1. Extension field format	6
3.2. Usage	6
4. Security option	6
4.1. Extension field format	7
4.2. Usage	7
4.3. Cookies	8
4.4. HMAC	8
4.4.1. Extension field format	8
4.4.2. Selecting a hash algorithm	9
4.4.3. Pre-shared key management	10
4.5. Interaction with other optional extensions	10
5. Fragmentation option	10
5.1. Motivation	11
5.2. Scope	12
5.3. Extension field format	12
5.4. Fragmentation procedure	13
5.5. Reassembly procedure	15
5.6. Security Considerations	16
6. Payload transform option	17
6.1. Extension field format	17
6.2. Usage	18
6.3. Interaction with other optional extensions	18
6.4. DTLS transform	19
7. Remote checksum offload option	19
7.1. Extension field format	20
7.2. Usage	20
7.2.1. Transmitter operation	20
7.2.2. Receiver operation	21
7.3. Security Considerations	22
8. Checksum option	22
8.1. Extension field format	22
8.2. Requirements	23
8.3. GUE checksum pseudo header	23
8.4. Usage	24

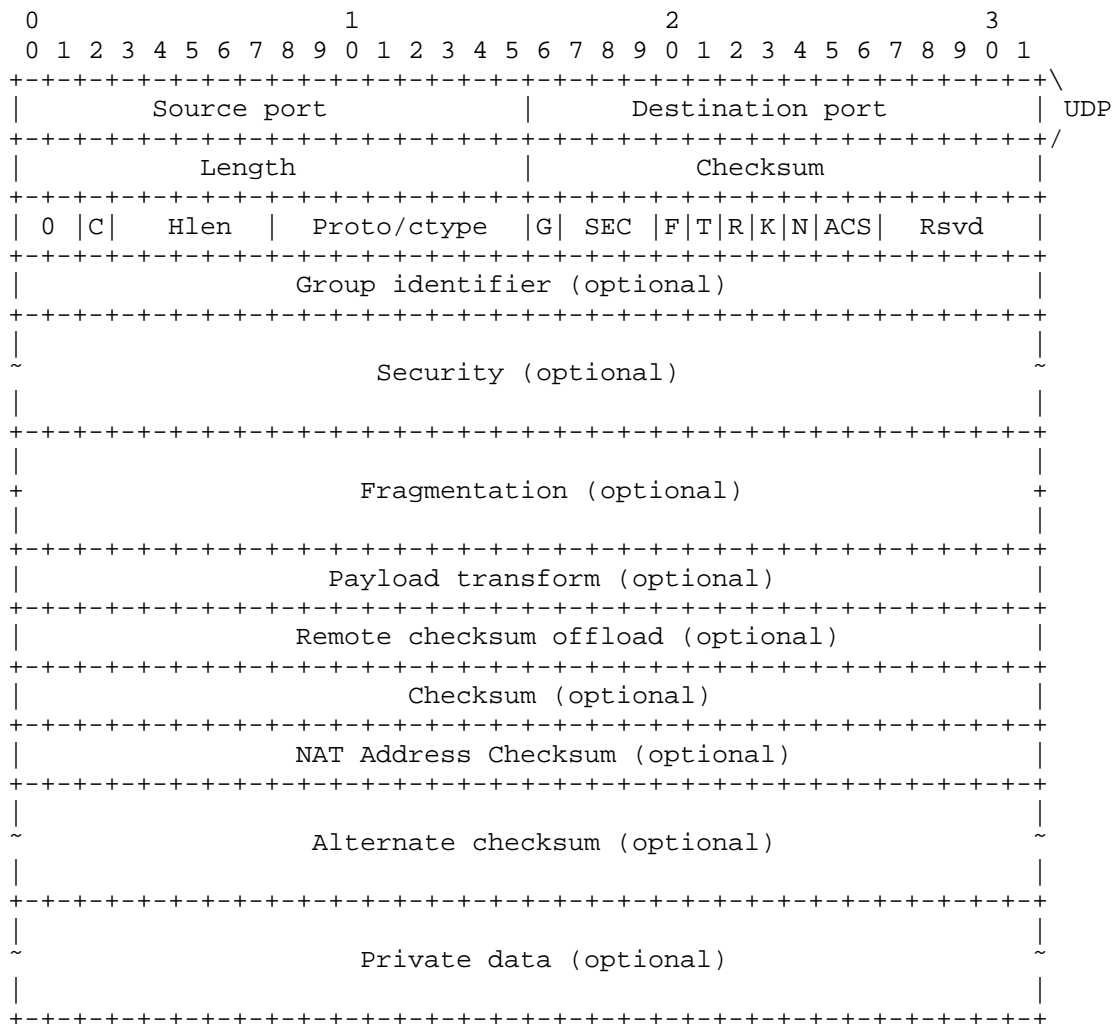
8.4.1. Transmitter operation	24
8.4.2. Receiver operation	25
8.5. Corrupted flags	25
8.6. Security Considerations	26
9. NAT checksum address option	26
9.1. Extension field format	26
9.2. Usage	26
9.2.1. Transmitter operation	27
9.2.2. Receiver operation	28
10. Alternative checksum option	28
10.1. Extension field format	28
10.1.1. CRC-16-CCITT and CRC-16 alternate checksums	29
10.1.2. 32-bit CRC alternate checksum	30
10.2. Usage	30
10.2.1. Transmitter operation	30
10.2.2. Receiver operation	31
10.3. Corrupted flags	31
10.4. Security Considerations	32
11. Processing order of options	32
11.1. Processing order when sending	32
11.2. Processing order when receiving	33
12. Security Considerations	33
13. IANA Consideration	34
14. References	36
14.1. Normative References	36
14.2. Informative References	36
Authors' Addresses	38

1. Introduction

Generic UDP Encapsulation (GUE) [I.D.ietf-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 group identifier, security, fragmentation, payload transform, remote checksum offload, checksum, NAT address checksum, and alternate checksum.

2. GUE header format with optional extensions

The format of a version 0 GUE header with optional extensions is:



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

The GUE header consists of:

- o Variant: Set to 0 to indicate GUE encapsulation header. Note that variant 1 (direct IP encapsulation) does not allow options.
- o C: C-bit. Indicates the GUE payload is a control message when set, a data message when not set. GUE optional extensions can be used with either control or data messages unless otherwise specified in the extension definition.
- o Hlen: Length in 32-bit words of the GUE header, including optional extension fields and private data but not the first four bytes of the header. Computed as $(\text{header_len} - 4) / 4$. The length of the encapsulated packet is determined from the UDP length and the Hlen: $\text{encapsulated_packet_length} = \text{UDP_Length} - 12 - 4 * \text{Hlen}$.
- o Proto/ctype: If the C-bit is not set this indicates IP protocol number for the packet in the payload; if the C bit is set this is the type of control message in the payload. The next header begins at the offset provided by Hlen. When the payload transform option or fragmentation option is used this field SHOULD be set to protocol number 59 for a data message, or zero for a control message, to indicate no next header for the payload.
- o G: Indicates the the group identifier extension field is present. The group identifier option is described in section 3.
- o SEC: Indicates security extension field is present. The security option is described in section 4.
- o F: Indicates fragmentation extension field is present. The fragmentation option is described in section 5.
- o T: Indicates payload transform extension field is present. The payload transform option is described in section 6.
- o R: Indicates the remote checksum extension field is present. The remote checksum offload option is described in section 7.
- o K: Indicates checksum extension field is present. The checksum option is described in section 8.
- o N: Indicates NAT address checksum field is present. The NAT address checksum option is described in section 9.

- o ACS: Indicates alternative checksum field is present. The alternative checksum option is described in section 10.
- o Private data is described in [I.D.ietf-gue].

3. Group identifier option

A group identifier classifies packets that logically belong to the same group. Groups are arbitrarily defined for different purposes and their definition is shared between the communicating end nodes.

3.1. Extension field format

The presence of the GUE group identifier option is indicated in the G flag bit of the GUE header.

The format of the group identifier option is:

```

      0                               1                               2                               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                               Group identifier                               |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

The fields of the option are:

- o Group identifier: Identifier value of a group.

3.2. Usage

The group identifier is set by an encapsulator to indicate that a packet belongs to a group. Groups may be arbitrarily defined to classify packets. Specific use cases of the group identifier may be defined in other documents ([I.D.hy-nvo3-gue-4-nvo] defines a use of this field to contain a virtual networking identifier for implementing network virtualization).

Intermediate nodes MAY apply semantics to group identifiers if group identifier information is shared and made global within a network. For instance, a firewall could block packets based on a group identifier that serves as a virtual identifier for a tenant.

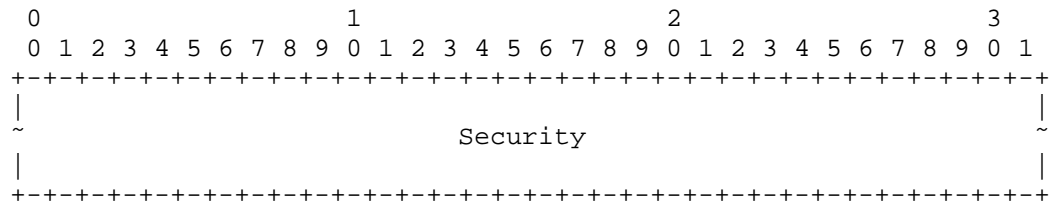
4. Security option

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.

4.1. 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:

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

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

The values in the SEC flags are:

- o 000b - No security field
- o 001b - 64 bit security field
- o 010b - 128 bit security field
- o 011b - 256 bit security field
- o 100b - 320 bit security field (HMAC)
- o 101b, 110b, 111b - Reserved values

4.2. Usage

The GUE security option is 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.

4.3. 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. A cookie is a shared value between an encapsulator and decapsulator which SHOULD be chosen randomly and MAY be changed periodically. Different cookies MAY be used for logical flows between the encapsulator and decapsulator; for instance packets sent with different VNIs in network virtualization [I.D.hy-nvo3-gue-4-nvo] might have different cookies. Cookies can be 64, 128, or 256 bits in size.

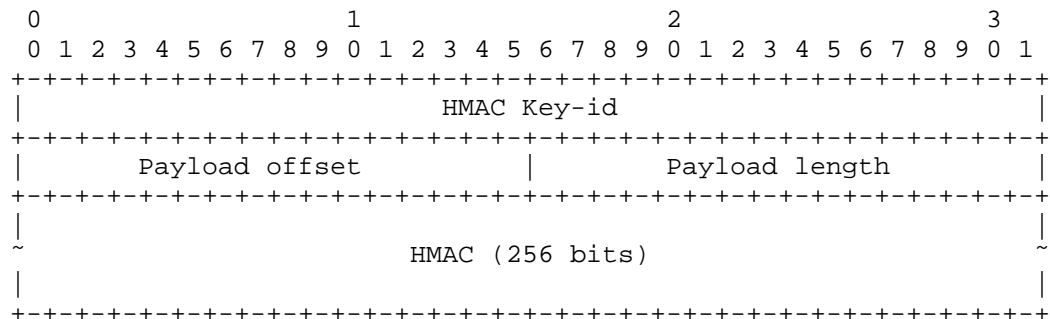
4.4. HMAC

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].

4.4.1. Extension field format

The HMAC option is a 320 bit field (40 octets). The security flags are set to 100b to indicate the presence of a 320 bit security field.

The format of the field is:



Fields are:

- o HMAC Key-id: opaque field to allow multiple hash algorithms or key selection
- o Payload offset: offset in payload that indicates the first octet of payload data included in the HMAC.

- o Payload length: length of payload data starting from the payload offset to be included in the HMAC calculation. Zero indicates no payload data is used in the calculation.
- o HMAC: Output of HMAC computation

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

- o The IP addresses
- o The GUE header including all optional extensions except for the security option
- o Optionally some or all of GUE payload. The portion of payload covered is indicated by an offset into the payload and a length relative to the offset.

The purpose of the HMAC option is to verify the validity, the integrity and the authentication of the GUE header itself and optionally some or all of the GUE payload.

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 when GUE endpoints converge to a fresher pre-shared key.

A portion of the GUE payload MAY be included in the HMAC calculation. The payload coverage is indicated in the option in the payload offset and payload length fields. If no payload data is included in the HMAC then the payload length field is set to zero. On reception, if the sum of the payload offset and the payload length is greater than the total length of the payload, then the packet MUST be dropped.

4.4.2. 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 bits, 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 bits 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.

4.4.3. Pre-shared key management

The field HMAC Key-id allows for:

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

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]

4.5. Interaction with other optional extensions

If GUE fragmentation (section 5) 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 6) 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 6.3 details handling for when both are used in a packet.

5. 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 [RFC8200]. Fragmentation can be performed on both data and control messages in GUE.

5.1. 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 of alternative #1 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. In this method, fragmentation is part of the encapsulation and an encapsulation header contains the information for reassembly. This differs from IP fragmentation in that the IP headers of the original packet are not replicated for each fragment.

Incorporating fragmentation into the encapsulation protocol has some advantages:

- o At least a 32 bit identifier can be defined to avoid issues of the 16 bit Identification in IPv4.
- o Encapsulation mechanisms for security and identification, such as group identifiers, can be applied to each segment.
- o This allows the possibility of using alternate fragmentation and reassembly algorithms (e.g. fragmentation with Forward Error Correction).
- o Fragmentation is transparent to the underlying network so it is unlikely that fragmented packet will be unconditionally dropped as might happen with IP fragmentation.

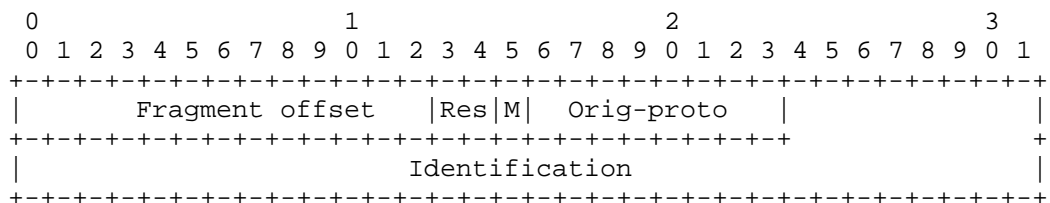
5.2. Scope

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.

5.3. 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 the C-bit in the GUE header is set) or the IP protocol (when C-bit is not set) of the fragmented packet.
- o Identification: 40 bits. Identifies fragments of a fragmented packet.

Pertinent GUE header fields to fragmentation are:

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

5.4. Fragmentation procedure

If an encapsulator determines that a packet must be fragmented (e.g. 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 group identifier if present.

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

```

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

```

Fragmentation and encapsulation are performed on the original packet in sequence. First the packet is divided up into 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:

```

+-----+-----+-----+--//--+-----+
| first | second | third |   | last |
| fragment | fragment | fragment | .... | fragment |
+-----+-----+-----+--//--+-----+

```

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

```

+-----+-----+-----+
| IP/UDP header | GUE header | first |
|               | w/ frag option | fragment |
+-----+-----+-----+

+-----+-----+-----+
| IP/UDP header | GUE header | second |
|               | w/ frag option | fragment |
+-----+-----+-----+

               o
               o

+-----+-----+-----+
| IP/UDP header | GUE header | last |
|               | w/ frag option | fragment |
+-----+-----+-----+

```

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 indicates the fragmentation option is present. The C-bit and and proto/ctype are set appropriately as described above.
- (3) The GUE fragmentation option. Orig-protocol is set to the protocol of the original packet. The M-bit is set for all fragments except the last one. Fragment offset is set as the offset of each fragment in the original packet.
- (4) Other GUE extensions.
- (5) The fragment itself as payload of the GUE packet.

5.5. Reassembly procedure

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

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

The following rules govern reassembly:

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

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

An original packet is reassembled only from GUE fragment packets that have the same outer source address, destination address, UDP source port, UDP destination port, GUE header C-bit, group 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 can 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.

5.6. Security Considerations

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

Attacks on GUE fragmentation can be mitigated by:

- o Hardened implementation that applies applicable techniques from implementation of IP fragmentation.

- o Application of GUE security (section 4) or IPsec [RFC4301]. Security mechanisms can prevent spoofing of fragments from unauthorized sources.
- o Implement fragment filter techniques for GUE encapsulation as described in [RFC1858] and [RFC3128].
- o Do not accept data in overlapping segments.
- o Enforce a minimum size for the first fragment.

6. 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.

6.1. 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:

0	1	2	3
0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9	0 1
+-----+-----+-----+-----+			
	Type		P_C_type
	Info		
+-----+-----+-----+-----+			

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 purposes or 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 is 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.

Info: 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.

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

6.3. Interaction with other optional extensions

If GUE fragmentation (section 5) 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.

6.4. 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 is interpreted as a DTLS record.

The payload transform option for DTLS is:

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|           1           | P_C_type |           0           |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

DTLS [RFC6347] provides a packet fragmentation capability. To avoid packet fragmentation being performed multiple times, a GUE encapsulator SHOULD use GUE fragmentation and not DTLS fragmentation.

DTLS usage 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 sessions with one or multiple tunnel encapsulators, and likewise a GUE tunnel encapsulator implementation can establish DTLS sessions with one or multiple decapsulators.

7. 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

- 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.

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

In pseudo code:

```
csum: initialized to checksum computed from start (outer IP
      header) to the end of the packet
start_of_packet: address of start of packet
encap_payload_offset: relative to start_of_packet
csum_start: value from the checksum start field
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: value from the checksum offset 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.

7.3. 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 **MUST** apply security checks on the GUE payload only after checksum remote offload has been processed.

8. Checksum option

The GUE checksum option provides a checksum that covers the GUE header, a GUE pseudo header, and optionally all or part 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 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.

8.1. 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:

0	1	2	3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1			
+-----+-----+-----+-----+			
	Checksum		Payload coverage
+-----+-----+-----+-----+			

The fields of the option are:

- o Checksum: Computed checksum value. This checksum covers the GUE header (including fields and private data covered by Hlen), the

GUE pseudo header, and optionally all or part of the payload (encapsulated packet).

- o Payload coverage: Number of bytes of payload to cover in the checksum. Zero indicates that the checksum only covers the GUE header and GUE pseudo header. If the value is greater than the encapsulated payload length, the packet **MUST** be dropped.

8.2. 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).

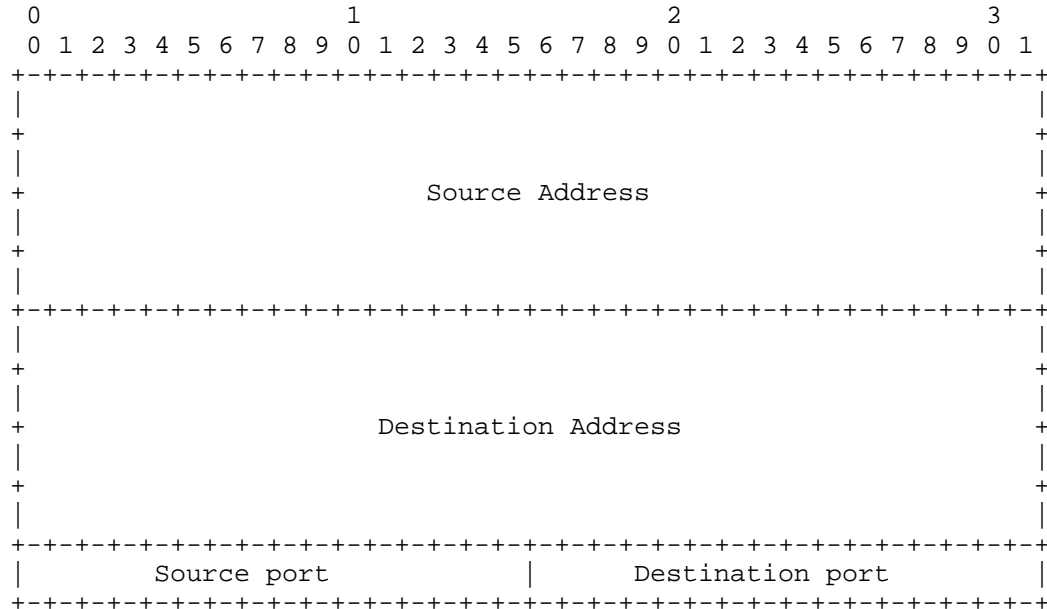
The GUE header checksum SHOULD NOT be set when the UDP checksum is non-zero. In this case the UDP checksum provides adequate protection and this avoids convolutions when a packet traverses NAT that does address translation (in that case the UDP checksum is required).

8.3. 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 [RFC8200] for IPv6.

The GUE pseudo header for IPv4 is:

The GUE pseudo header for IPv6 is:



The GUE pseudo header does not include payload length or protocol as in the standard IP pseudo headers. The length field is deemed unnecessary for inclusion because a corrupted length field should not cause mis-delivery, the GUE checksum is applied after GUE fragmentation, and without the length field the GUE pseudo header checksum is the same for all packets of flow.

8.4. 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.

8.4.1. 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.
- 6) Set the bitwise not of the resultant value in the GUE checksum field.

8.4.2. 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 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.

8.5. Corrupted flags

Note that the GUE checksum does not protect against the checksum flag (K flag) being corrupted. If an encapsulator sets the checksum flag and option but the K bit flips to be zero, then a decapsulator will incorrectly process the GUE packet as not having a checksum field.

To mitigate this issue an encapsulator and decapsulator might agree

that checksum is always required. This agreement could be established by configuration or GUE capability negotiation.

8.6. 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.

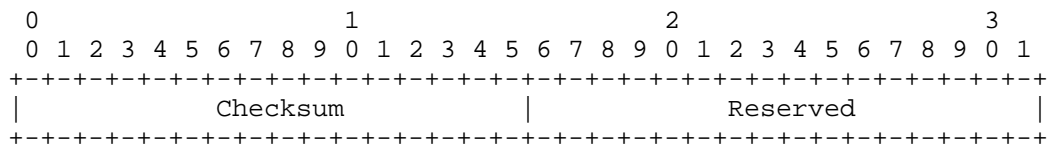
9. NAT checksum address option

The NAT address checksum (NAC) option contains the checksum computed over the IP addresses of the packet. The computed value can be used by a receiver to adjust a transport layer checksum that was affected by NAT changing the IP addresses. This option is only useful when GUE encapsulates a transport layer packet that has a checksum with a pseudo header that includes the IP addresses (such as TCP or UDP).

9.1. Extension field format

The presence of the NAT checksum address option is indicated by the N bit in the GUE header.

The format of the NAT checksum address extension is:



The fields of the option are:

- o Checksum: Computed checksum value. This checksum covers the outer IP addresses.
- o Reserved: Must be zero on transmit.

9.2. Usage

The NAT address extension SHOULD be set on transmit when encapsulating a transport layer packet whose checksum might be affected by NAT being performed on the outer IP header. If this option is used then the UDP checksum MUST be used (sent with non-zero value).

The NAT address checksum is computed using the Internet checksum

[RFC1071].

9.2.1.1. Transmitter operation

The procedure for setting the GUE checksum on transmit is:

An encapsulator computes the checksum value over the IP addresses in the IP header.

- 1) Compute the ones complement checksum over the source and destination IPv4 or IPv6 addresses.
- 2) Set the resultant value in the Checksum field.

For IPv4 the checksum is computed over:

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     Source Address                                     |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     Destination Address                               |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

For IPv6 the checksum is computed over:

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     Source Address                                     |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     Destination Address                               |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

9.2.2. Receiver operation

- 1) Validate the UDP checksum is correct
- 2) Compute the checksum over the IP address in the received packet
- 3) Subtract the resultant from the checksum value in the NAC option. If the difference is non-zero then NAT has changed the addresses
- 4) When processing a transport layer containing a checksum affected by NAT on the IP addresses, add the difference into the checksum calculation when verify the packet.

In pseudo codes this is:

```
actual = checksum(IP addresses)
diff = actual - NAC_value
verify = checksum(transport packet) + checksum(pseudo header)
        + diff

if (verify == 0)
    packet is good
```

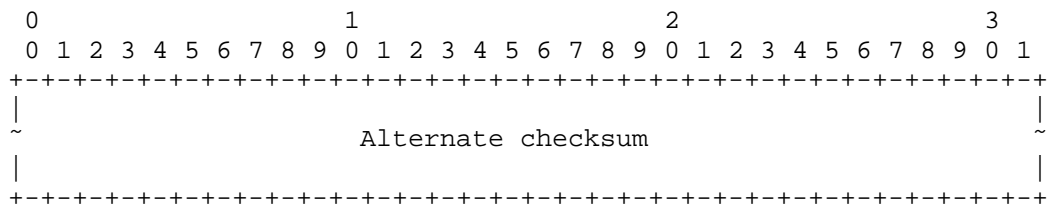
10. Alternative checksum option

The alternative checksum option contains a check over the GUE header and optionally all or part of the GUE payload. The alternative checksum can provide stronger protection against data corruption than provided by the one's complement checksum used in the UDP checksum or the GUE checksum (section 8).

The alternative checksum flags are two bits which allow three methods to be defined. This specification proposes three: two 16-bit CRC variants and a 32-bit CRC method.

10.1. Extension field format

The format of the alternate checksum option is:



The fields of the option are:

- o Alternate checksum (variable length). Contains the checksum information.

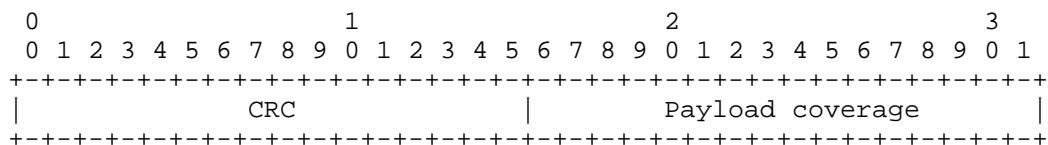
The presence of the alternate checksum option is indicated by the ACS bits in the GUE header.

The values in the ACS flags are:

- o 00b - No alternate checksum field
- o 01b - CRC-16-CCITT
- o 10b - CRC-16
- o 11b - CRC-32

10.1.1.1. CRC-16-CCITT and CRC-16 alternate checksums

CRC-16-CCITT and CRC-16 have the same alternative checksum field format. The format is:



The fields of the option are:

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

The 16-bit alternative checksums do not include a pseudo header containing IP addresses or ports.

CRC-16-CCITT is calculated using the polynomial:

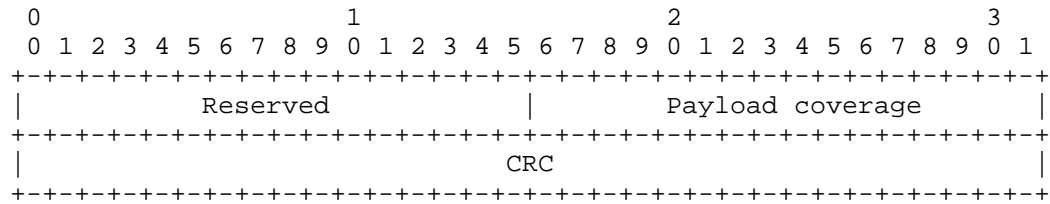
$$x^{16} + x^{12} + x^5 + 1 \text{ (polynomial representation 0x1021).}$$

CRC-16 is calculated using the polynomial:

$x^{16} + x^{15} + x^2 + 1$ (polynomial representation 0x8005).

10.1.2. 32-bit CRC alternate checksum

The format of the 32-bit CRC alternative checksum field is:



The fields of the option are:

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

The 32-bit alternative checksum does not include a pseudo header containing IP addresses or ports.

CRC-32 is calculated using the polynomial:

$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{10} + x^2 + x^7 + x^5 + x^4 + x^2 + x + 1$ (polynomial 0x04C11DB7).

10.2. Usage

The usage of the alternate checksum is the same for both the 16-bit CRC and the 32-bit CRC methods except that the output CRC values have different sizes.

10.2.1. Transmitter operation

The procedure for setting the GUE alternative checksum on transmit is:

- 1) Create the GUE header including the alternative checksum field. The CRC field is initialized to zero.

- 2) Calculate the alternative checksum (either a 16-bit or 32-bit CRC) from the start of the GUE header through the its length as indicated in GUE Hlen.
- 3) Continue the calculation to cover the payload portion if payload coverage is enabled (payload coverage field is non-zero). If the length of the payload coverage is not aligned per the algorithm (four bytes for CRC-32 and two bytes for CRC-16), logically append zero bytes up to the necessary alignment for the purposes of CRC calculation.
- 4) Set the resultant value in the CRC field.

10.2.2. Receiver operation

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

The procedure for verifying the alternate checksum is:

- 1) If the payload coverage length is greater than the length of the encapsulated payload then drop the packet.
- 2) Note value in the CRC field and set the CRC field to zero.
- 3) Calculate the alternative checksum (either a 16-bit or 32-bit CRC) from the start of the GUE header through the its length as indicated in GUE Hlen.
- 4) Continue the calculation to cover the payload portion if payload coverage is enabled (payload coverage field is non-zero). If the length of the payload coverage is not aligned per the algorithm (four bytes for CRC-32 and two bytes for CRC-16), logically append zero bytes up to the necessary alignment for the purposes of CRC calculation.
- 5) Compare the computed value with the original value of the CRC field. If they are equal then that packet is accepted, if they are not equal then verification failed and the packet MUST be dropped.
- 6) Restore the CRC field to its original value.

10.3. Corrupted flags

Similar to GUE checksum properties, the GUE alternate checksum does not protect against the alternative checksum flags (ACS flags) being

corrupted. If an encapsulator sets the alternative checksum flags and option but the ACS bits flips to be zero, then a decapsulator will incorrectly process that packet as not having an alternate checksum field.

To mitigate this issue an encapsulator and decapsulator might agree that an alternate checksum is always required. This agreement could be established by configuration or GUE capability negotiation.

10.4. Security Considerations

The alternate 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.

11. Processing order of options

Options MUST be processed in a specific order for both transmission and reception. Note that some options, such as the checksum option, depend on other fields in the GUE header to be initialized.

11.1. Processing order when sending

When setting the security option (HMAC option in particular), the checksum option, and the alternate checksum option-- all the GUE fields being used must be present and properly set in the header. The checksum value in the checksum option or alternate checksum option MUST be initialized to zero to ensure consistent HMAC and checksum calculation.

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

- 1) Fragment if necessary and set fragmentation option. If the group identifier is present it is copied into each fragment. If payload transformation will increase the size of the payload that MUST be accounted for when deciding how to fragment. Apply processing below for each fragment
- 2) Set group identifier option (to the same value for each fragment)
- 3) Perform payload transform (potentially on a fragment) and set payload transform option.
- 4) Set Remote checksum offload.
- 5) Set NAT address checksum option.

- 6) Set security option per cookie or HMAC calculation.
- 7) Calculate GUE checksum and set checksum option.
- 8) Calculate GUE alternate checksum and set the alternate checksum option.

11.2. Processing order when receiving

On reception the order of actions is:

- 1) Verify GUE alternate checksum.
- 2) Verify GUE checksum. If the alternate checksum option is present, set its checksum fields to zero for computing the GUE checksum. After computation, restore the checksum value in the alternate checksum field.
- 3) Verify security option. If the GUE checksum option or alternate checksum option are also present and HMAC computation is being done over the GUE header, then set the checksum fields to zero for computing the HMAC. After computation, restore the checksum values.
- 4) Save the NAT address checksum value. It will be applied when processing the encapsulated packet.
- 5) Adjust packet for remote checksum offload.
- 6) Perform payload transformation (i.e. decrypt payload).
- 7) Perform reassembly.
- 8) Process packet (take group identifier into account if present).

The relative processing order of GUE extensions and private fields is unspecified in this specification.

12. Security Considerations

Encapsulation of a network protocol in GUE should not increase security risk, nor provide additional security in itself. GUE requires that the source port for UDP packets SHOULD be randomly seeded to mitigate some possible denial service attacks.

If the integrity and privacy of data packets being transported through GUE is a concern, GUE security option and payload encryption using 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 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. This is a drawback since it 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 for application level 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.

13. IANA Consideration

IANA is requested to assign flags for the extensions defined in this specification. Specifically, an assignment is requested for the Group Identifier, Security, Fragmentation, Payload Transform, Remote Checksum Offload, Checksum, NAT Checksum, and Alternate Checksum extensions in the "GUE flag-fields" registry.

Flags bits	Field size	Description	Reference
Bit 0	4 bytes	Group identifier	This document
Bit 1..3	001->8 bytes 010->16 bytes 011->32 bytes 100->40 bytes	Security	This document
Bit 4	8 bytes	Fragmen- tation	This document
Bit 5	4 bytes	Payload transform	This document
Bit 6	4 bytes	Remote checksum offload	This document
Bit 7	4 bytes	Checksum	This document
Bit 8	4 bytes	NAT checksum address	This document
Bit 9..10	01->4 bytes 10->8 bytes	Alternate checksum	This document
Bit 11..15		Unassigned	

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

14. References

14.1. Normative References

- [RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.
- [RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <<https://www.rfc-editor.org/info/rfc8200>>.
- [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.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <<http://www.rfc-editor.org/info/rfc5226>>.
- [I.D.ietf-gue] T. Herbert, L. Yong, and O. Zia, "Generic UDP Encapsulation" draft-ietf-intarea-gue-01

14.2. Informative References

- [RFC3931] Lau, J., Townsley, W., et al, "Layer Two Tunneling Protocol - Version 3 (L2TPv3)", RFC3931, 1999
- [RFC2104] Kent, S. and R. Atkinson, "Security Architecture for the Internet Protocol" , RFC 2401, DOI 10.17487/RFC2401, November 1998, <<http://www.rfc-editor.org/info/rfc2401>>.
- [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>>.
- [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>>.
- [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.
- [RFC6347] Rescoria, E., Modadugu, N., "Datagram Transport Layer Security Version 1.2", RFC6347, 2012.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<http://www.rfc-editor.org/info/rfc793>>.
- [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>>.
- [RFC1071] Braden, R., Borman, D., and C. Partridge, "Computing the Internet checksum", RFC1071, September 1988.
- [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.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
- [UDPENCAP] T. Herbert, "UDP Encapsulation in Linux", <http://people.netfilter.org/pablo/netdev0.1/papers/UDP-Encapsulation-in-Linux.pdf>
- [FIPS180-4] Secure Hash Standard (SHS), Nation Institute of Standards and Technology, 8/2015

Authors' Addresses

Tom Herbert
Quantonium
4701 Patrick Henry Dr.
Santa Clara, 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

intarea
Internet-Draft
Intended status: Standards Track
Expires: August 13, 2018

P. Pfister
E. Vyncke, Ed.
Cisco
T. Pauly
D. Schinazi
Apple
February 9, 2018

Discovering Provisioning Domain Names and Data
draft-ietf-intarea-provisioning-domains-01

Abstract

An increasing number of hosts access the Internet via multiple interfaces or, in IPv6 multi-homed networks, via multiple IPv6 prefix configurations.

This document describes a way for hosts to identify such means, called Provisioning Domains (PvDs), with Fully Qualified Domain Names (FQDN). Those identifiers are advertised in a new Router Advertisement (RA) option and, when present, are associated with the set of information included within the RA.

Based on this FQDN, hosts can retrieve additional information about their network access characteristics via an HTTP over TLS query. This allows applications to select which Provisioning Domains to use as well as to provide configuration parameters to the transport layer and above.

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 <https://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 August 13, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
2. Terminology	3
3. Provisioning Domain Identification using Router Advertisements	4
3.1. PvD ID Option for Router Advertisements	4
3.2. Router Behavior	7
3.3. Host Behavior	7
3.3.1. DHCPv6 configuration association	8
3.3.2. DHCPv4 configuration association	8
3.3.3. Interconnection Sharing by the Host	9
4. Provisioning Domain Additional Information	9
4.1. Retrieving the PvD Additional Information	9
4.2. Operational Consideration to Providing the PvD Additional Information	10
4.3. PvD Additional Information Format	11
4.3.1. Private Extensions	12
4.3.2. Example	12
4.4. Detecting misconfiguration and misuse	13
5. Operation Considerations	13
6. Security Considerations	15
7. Privacy Considerations	15
8. IANA Considerations	16
9. Acknowledgements	16
10. References	16
10.1. Normative references	16
10.2. Informative references	17
Appendix A. Changelog	19
A.1. Version 00	19
A.2. Version 01	19
A.3. Version 02	20
A.4. WG Document version 00	20

A.5. WG Document version 01	21
Authors' Addresses	21

1. Introduction

It has become very common in modern networks for hosts to access the internet through different network interfaces, tunnels, or next-hop routers. To describe the set of network configurations associated with each access method, the concept of Provisioning Domain (PvD) was defined in [RFC7556].

This document specifies a way to identify PvDs with Fully Qualified Domain Names (FQDN), called PvD IDs. Those identifiers are advertised in a new Router Advertisement (RA) [RFC4861] option called the PvD ID Router Advertisement option which, when present, associates the PvD ID with all the information present in the Router Advertisement as well as any configuration object, such as addresses, deriving from it. The PVD ID Router Advertisement option may also contain a set of other RA options. Since such options are only considered by hosts implementing this specification, network operators may configure hosts that are 'PvD-aware' with PvDs that are ignored by other hosts.

Since PvD IDs are used to identify different ways to access the internet, multiple PvDs (with different PvD IDs) could be provisioned on a single host interface. Similarly, the same PvD ID could be used on different interfaces of a host in order to inform that those PvDs ultimately provide identical services.

This document also introduces a way for hosts to retrieve additional information related to a specific PvD by means of an HTTP over TLS query using an URI derived from the PvD ID. The retrieved JSON object contains additional information that would typically be considered unfit, or too large, to be directly included in the Router Advertisement, but might be considered useful to the applications, or even sometimes users, when choosing which PvD and transport should be used.

2. Terminology

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 [RFC2119].

In addition, this document uses the following terminology:

Provisioning Domain (PvD): A set of network configuration information; for more information, see [RFC7556].

PvD ID: A Fully Qualified Domain Name (FQDN) used to identify a PvD.

Explicit PvD: A PvD uniquely identified with a PvD ID. For more information, see [RFC7556].

Implicit PvD: A PvD that, in the absence of a PvD ID, is identified by the host interface to which it is attached and the address of the advertising router.

3. Provisioning Domain Identification using Router Advertisements

Explicit PvDs are identified by a PvD ID. The PvD ID is a Fully Qualified Domain Name (FQDN) which MUST belong to the network operator in order to avoid naming collisions. The same PvD ID MAY be used in several access networks when they ultimately provide identical services (e.g., in all home networks subscribed to the same service).

3.1. PvD ID Option for Router Advertisements

This document introduces a Router Advertisement (RA) option called PvD ID Router Advertisement option. It is used to convey the FQDN identifying a given PvD (see Figure 1), bind the PvD ID with configuration information received over DHCPv4 (see Section 3.3.2), enable the use of HTTP over TLS to retrieve the PvD Additional Information JSON object (see Section 4), as well as contain any other RA options which would otherwise be valid in the RA.

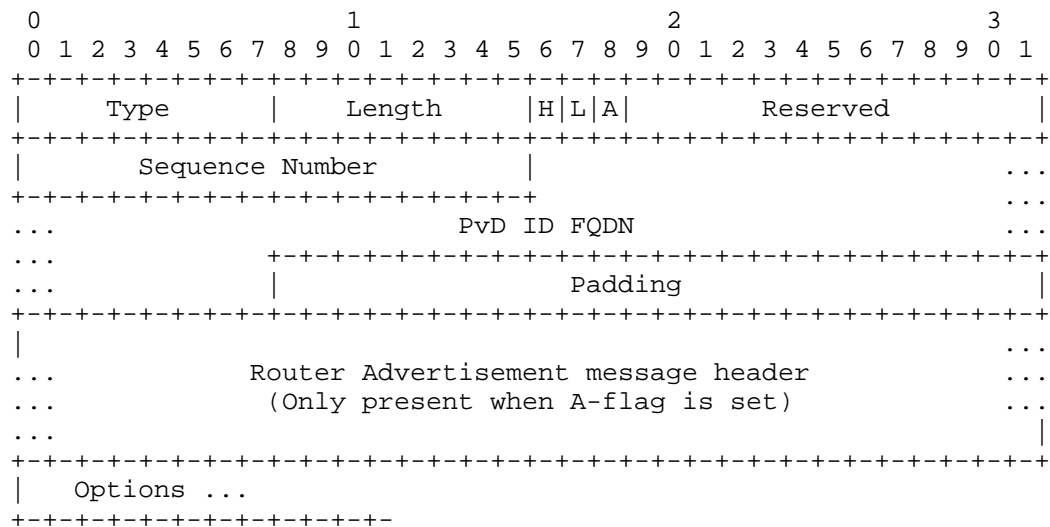


Figure 1: PvD ID Router Advertisements Option format

- Type : (8 bits) To be defined by IANA. Current experimentation uses the value of 254.
- Length : (8 bits) The length of the option in units of 8 octets, including the Type and Length fields, the Router Advertisement message header, if any, as well as the RA options that are included within the PvD ID Option.
- H-flag : (1 bit) 'HTTP' flag stating whether some PvD Additional Information is made available through HTTP over TLS, as described in Section 4.
- L-flag : (1 bit) 'Legacy' flag stating whether the router is also providing IPv4 information using DHCPv4 (see Section 3.3.2).
- A-flag : (1 bit) 'Advertisement' flag stating whether the PvD ID Option is followed (right after padding to the next 64 bits boundary) by a Router Advertisement message header (See section 4.2 of target="RFC4861"/>).
- Reserved : (13 bits) Reserved for later use. It MUST be set to zero by the sender and ignored by the receiver.
- Sequence Number: (16 bits) Sequence number for the PvD Additional Information, as described in Section 4.

PvD ID FQDN : The FQDN used as PvD ID encoded in DNS format, as described in Section 3.1 of [RFC1035]. Domain names compression described in Section 4.1.4 of [RFC1035] MUST NOT be used.

Padding : Zero or more padding octets to the next 8 octets boundary. It MUST be set to zero by the sender, and ignored by the receiver.

RA message header : (16 octets) When the A-flag is set, a full Router Advertisement message header as specified in [RFC4861]. The 'Type', 'Code' and 'Checksum' fields (i.e. the first 32 bits), MUST be set to zero by the sender and ignored by the receiver. The other fields are to be set and parsed as specified in [RFC4861] or any updating documents.

Options : Zero or more RA options that would otherwise be valid as part of the Router Advertisement main body, but are instead included in the PvD ID Option such as to be ignored by hosts that are not 'PvD-aware'.

Here is an example of a PvD ID option with example.org as the PvD ID FQDN and including a RDNSS and prefix information options:

0										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1								
Type: 254										Length: 12										0 0 0										Reserved									
Sequence Number										7										e																			
m										a										m										p									
l										e										3										o									
r										g										0										0 (padding)									
0 (padding)										0 (padding)										0 (padding)										0 (padding)									
RDNSS option (RFC 6106) length: 5																														...									
...																														...									
Prefix Information Option (RFC 4861) length: 4																														...									
...																														...									
...																														...									

3.2. Router Behavior

A router MAY send RAs containing at most one PvD ID RA option, but MUST NOT include more than one PvD ID RA option in each RA. In particular, the PvD ID RA option MUST NOT contain further PvD ID RA options.

The PvD ID Option MAY contain zero, one, or more RA options which would otherwise be valid as part of the same RA. Such options are processed by PvD-aware hosts, while ignored by others.

In order to provide multiple different PvDs, a router MUST send multiple RAs. Different explicit PvDs MAY be advertised with RAs using the same IPv6 source address; but different implicit PvDs, advertised with different RAs, MUST use different link local addresses.

Whenever an RA, for a single PvD, would need to be sent via multiple packets, the PvD ID RA option header (i.e., all fields except the 'Options' field) MUST be repeated in all the transmitted RAs. But the options within the 'Options' field, MAY be transmitted only once, included in one of the transmitted PvD ID RA options.

3.3. Host Behavior

Hosts MUST associate received RAs and included configuration information (e.g., Router Valid Lifetime, Prefix Information [RFC4861], Recursive DNS Server [RFC8106], Routing Information [RFC4191] options) with the explicit PvD identified by the first PvD ID Option present in the received RA, if any, or with the implicit PvD identified by the host interface and the source address of the received RA otherwise.

In case multiple PvD ID options are found in a given RA, hosts MUST ignore all but the first PvD ID option.

Similarly, hosts MUST associate all network configuration objects (e.g., default routers, addresses, more specific routes, DNS Recursive Resolvers) with the PvD associated with the RA which last updated the object. For example, addresses that are generated using a received Prefix Information option (PIO) are associated with the PvD of the last received RA which included the given PIO.

PvD IDs MUST be compared in a case-insensitive manner (i.e., A=a), assuming ASCII with zero parity while non-alphabetic codes must match exactly (see also Section 3.1 of [RFC1035]). For example, pvd.example.com or PvD.Example.coM would refer to the same PvD.

While resolving names, executing the default address selection algorithm [RFC6724] or executing the default router selection algorithm ([RFC2461], [RFC4191] and [RFC8028]), hosts MAY consider only the configuration associated with an arbitrary set of PvDs.

For example, a host MAY associate a given process with a specific PvD, or a specific set of PvDs, while associating another process with another PvD. A PvD-aware application might also be able to select, on a per-connection basis, which PvDs should be used. In particular, constrained devices such as small battery operated devices (e.g. IoT), or devices with limited CPU or memory resources may purposefully use a single PvD while ignoring some received RAs containing different PvD IDs.

The way an application expresses its desire to use a given PvD, or a set of PvDs, or the way this selection is enforced, is out of the scope of this document. Useful insights about these considerations can be found in [I-D.kline-mif-mpvd-api-reqs].

3.3.1. DHCPv6 configuration association

When a host retrieves configuration elements using DHCPv6 (e.g., addresses or DNS recursive resolvers), they MUST be associated with the explicit or implicit PvD of the RA received on the same interface, sent from the same LLA, and with the O-flag or M-flag set [RFC4861]. If no such PvD is found, or whenever multiple different PvDs are found, the host behavior is unspecified.

This process requires hosts to keep track of received RAs, associated PvD IDs, and routers LLA; it also assumes that the router either acts as a DHCPv6 server or relay and uses the same LLA for DHCPv6 and RA traffic (which may not be the case when the router uses VRRP to send its RA).

3.3.2. DHCPv4 configuration association

When a host retrieves configuration elements from DHCPv4, they MUST be associated with the explicit PvD received on the same interface, whose PVD ID Options L-flag is set and, in the case of a non point-to-point link, using the same datalink address. If no such PvD is found, or whenever multiple different PvDs are found, the configuration elements coming from DHCPv4 MUST be associated with the implicit PvD identified by the interface on which the DHCPv4 transaction happened. The case of multiple explicit PvD for an IPv4 interface is undefined.

3.3.3. Interconnection Sharing by the Host

The situation when a node receives an RA on one interface (e.g. cellular) and shares this connectivity by also acting as a router by transmitting RA on another interface (e.g. WiFi) is known as 'tethering'. It can be done as ND proxy. The exact behavior is TBD but it is expected that the one or several PvD associated to the shared interface (e.g. cellular) will also be advertised to the clients on the other interface (e.g. WiFi).

4. Provisioning Domain Additional Information

Additional information about the network characteristics can be retrieved based on the PvD ID. This set of information is called PvD Additional Information, and is encoded as a JSON object [RFC7159].

The purpose of this additional set of information is to securely provide additional information to applications about the connectivity that is provided using a given interface and source address pair. It typically includes data that would be considered too large, or not critical enough, to be provided within an RA option. The information contained in this object MAY be used by the operating system, network libraries, applications, or users, in order to decide which set of PvDs should be used for which connection, as described in Section 3.3.

4.1. Retrieving the PvD Additional Information

When the H-flag of the PvD ID Option is set, hosts MAY attempt to retrieve the PvD Additional Information associated with a given PvD by performing an HTTP over TLS [RFC2818] GET query to `https://<PvD-ID>/well-known/pvd` [RFC5785]. Inversely, hosts MUST NOT do so whenever the H-flag is not set.

Note that the DNS name resolution of the PvD ID, the PKI checks as well as the actual query MUST be performed using the considered PvD. In other words, the name resolution, PKI checks, source address selection, as well as the next-hop router selection MUST be performed while using exclusively the set of configuration information attached with the PvD, as defined in Section 3.3. In some cases, it may therefore be necessary to wait for an address to be available for use (e.g., once the Duplicate Address Detection or DHCPv6 processes are complete) before initiating the HTTP over TLS query. If the PvD allows for temporary address per [RFC4941], then hosts SHOULD use a temporary address to fetch the PvD Additional Information and SHOULD deprecate the used temporary address and generate a new temporary address afterward.

If the HTTP status of the answer is greater than or equal to 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 399, inclusive, it MUST follow the redirection(s). If the HTTP status of the answer is between 200 and 299, inclusive, the host MAY get a file containing a single JSON object. When a JSON object could not be retrieved, an error message SHOULD be logged and/or displayed in a rate-limited fashion.

After retrieval of the PvD Additional Information, hosts MUST keep track of the Sequence Number value received in subsequent RAs including the same PvD ID. In case the new value is greater than the value that was observed when the PvD Additional Information object was retrieved (using serial number arithmetic comparisons [RFC1982]), or whenever the validity time included in the PvD Additional Information JSON object is expired, hosts MUST either perform a new query and retrieve a new version of the object, or, failing that, deprecate the object and stop using the additional information provided in the JSON object.

Hosts retrieving a new PvD Additional Information object MUST check for the presence and validity of the mandatory fields specified in Section 4.3. A retrieved object including an expiration time that is already past or missing a mandatory element MUST be ignored. In order to avoid synchronized queries toward the server hosting the PvD Additional Information when an object expires, a host which last retrieved an object at a time A, including a validity time B, SHOULD renew the object at a uniformly random time in the interval $[(B-A)/2, A]$.

The PvD Additional Information object includes a set of IPv6 prefixes (under the key "prefixes") which MUST be checked against all the Prefix Information Options advertised in the RA. If any of the prefixes included in the PIO is not covered by at least one of the listed prefixes, the PvD associated with the tested prefix MUST be considered unsafe and MUST NOT be used. While this does not prevent a malicious network provider, it does complicate some attack scenarios, and may help detecting misconfiguration.

4.2. Operational Consideration to Providing the PvD Additional Information

Whenever the H-flag is set in the PvD RA Option, a valid PvD Additional Information object MUST be made available to all hosts receiving the RA by the network operator. In particular, when a captive portal is present, hosts MUST still be allowed to perform DNS, PKI and HTTP over TLS operations related to the retrieval of the object, even before logging into the captive portal.

Routers MAY increment the PVD ID Sequence number in order to inform host that a new PvD Additional Information object is available and should be retrieved.

The server providing the JSON files SHOULD also check whether the client address is part of the prefixes listed into the additional information and SHOULD return a 403 response code if there is no match.

4.3. PvD Additional Information Format

The PvD Additional Information is a JSON object.

The following table presents the mandatory keys which MUST be included in the object:

JSON key	Description	Type	Example
name	Human-readable service name	UTF-8 string [RFC3629]	"Awesome Wifi"
expires	Date after which this object is not valid	[RFC3339]	"2017-07-23T06:00:00Z"
prefixes	Array of IPv6 prefixes valid for this PVD	Array of strings	["2001:db8:1::/48", "2001:db8:4::/48"]

A retrieved object which does not include a valid string associated with the "name" key at the root of the object, or a valid date associated with the "expires" key, also at the root of the object, MUST be ignored. In such cases, an error message SHOULD be logged and/or displayed in a rate-limited fashion. If the PIO of the received RA is not covered by at least one of the "prefixes" key, the retrieved object SHOULD be ignored.

The following table presents some optional keys which MAY be included in the object.

JSON key	Description	Type	Example
localizedName	Localized user-visible service name, language can be selected based on the HTTP Accept-Language header in the request.	UTF-8 string	"Wifi Genial"
dnsZones	DNS zones searchable and accessible	array of DNS zones	["example.com", "sub.example.org"]
noInternet	No Internet, set when the PvD only provides restricted access to a set of services	boolean	true

It is worth noting that the JSON format allows for extensions. Whenever an unknown key is encountered, it MUST be ignored along with its associated elements.

4.3.1. Private Extensions

JSON keys starting with "x-" are reserved for private use and can be utilized to provide information that is specific to vendor, user or enterprise. 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.

4.3.2. Example

Here are two examples based on the keys defined in this section.

```
{
  "name": "Foo Wireless",
  "localizedName": "Foo-France Wifi",
  "expires": "2017-07-23T06:00:00Z",
  "prefixes" : ["2001:db8:1::/48", "2001:db8:4::/48"],
}
```

```
{
  "name": "Bar 4G",
  "localizedName": "Bar US 4G",
  "expires": "2017-07-23T06:00:00Z",
  "prefixes": ["2001:db8:1::/48", "2001:db8:4::/48"],
}
```

4.4. Detecting misconfiguration and misuse

When a host retrieves the PvD Additional Information, it MUST verify that the TLS server certificate is valid for the performed request (e.g., that the Subject Name is equal to the PvD ID expressed as an FQDN). This authentication creates a secure binding between the information provided by the trusted Router Advertisement, and the HTTPS server. But this does not mean the Advertising Router and the PvD server belong to the same entity.

Hosts MUST verify that all prefixes in the RA PIO are covered by a prefix from the PvD Additional Information. An adversarial router willing to fake the use of a given explicit PvD, without any access to the actual PvD Additional Information, would need to perform NAT66 in order to circumvent this check.

It is also RECOMMENDED that the HTTPS server checks the source addresses of incoming connections (see Section 4.1). This check give reasonable assurance that neither NPTv6 [RFC6296] nor NAT66 were used and restricts the information to the valid network users.

5. Operation Considerations

This section describes some use cases of PvD. For sake of simplicity, the RA messages will not be described in the usual ASCII art but rather in an indented list. For example, a RA message containing some options and a PvD ID option that also contains other options will be described as:

- o RA Header: router lifetime = 6000
- o Prefix Information Option: length = 4, prefix = 2001:db8:cafe::/64
- o PvD ID header: length = 3+ 5 +4 , PvD ID FQDN = example.org, A-flag = 0 (actual length of the header with padding 24 bytes = 3 * 8 bytes)
 - * Recursive DNS Server: length = 5, addresses= [2001:db8:cafe::53, 2001:db8:f00d::53]

- * Prefix Information Option: length = 4, prefix = 2001:db8:f00d::/64

It is expected that for some years, networks will have a mix of PvD-aware hosts and PvD-ignorant hosts. If there is a need to give specific information to PvD-aware hosts only, then it is recommended to send TWO RA messages: one for each class of hosts. For example, here is the RA for PvD-ignorant hosts:

- o RA Header: router lifetime = 6000 (PvD-ignorant hosts will use this router as a default router)
- o Prefix Information Option: length = 4, prefix = 2001:db8:cafe::/64
- o Recursive DNS Server Option: length = 3, addresses= [2001:db8:cafe::53]
- o PvD ID header: length = 3+ 2, PvD ID FQDN = foo.example.org, A-flag = 1 (actual length of the header 24 bytes = 3 * 8 bytes)
- * RA Header: router lifetime = 0 (PvD-aware hosts will not use this router as a default router), implicit length = 2

And here is a RA example for PvD-aware hosts:

- o RA Header: router lifetime = 0 (PvD-ignorant hosts will not use this router as a default router)
- o PvD ID header: length = 3+ 2 + 4 + 3, PvD ID FQDN = example.org, A-flag = 1 (actual length of the header 24 bytes = 3 * 8 bytes)
- * RA Header: router lifetime = 1600 (PvD-aware hosts will use this router as a default router), implicit length = 2
- * Prefix Information Option: length = 4, prefix = 2001:db8:f00d::/64
- * Recursive DNS Server Option: length = 3, addresses= [2001:db8:f00d::53]

In the above example, PvD-ignorant hosts will only use the first RA sent from their default router and using the 2001:db8:cafe::/64 prefix. PvD-aware hosts will autonomously configure addresses from both PIOs, but will only use the source address in 2001:db8:f00d::/64 to communicate past the first hop router since only the router sending the second RA will be used as default router; similarly, they will use the DNS server 2001:db8:f00d::53 when communicating with this address.

6. Security Considerations

Although some solutions such as IPsec or SeND [RFC3971] can be used in order to secure the IPv6 Neighbor Discovery Protocol, actual deployments largely rely on link layer or physical layer security mechanisms (e.g. 802.1x [IEEE8021X]) in conjunction with RA Guard [RFC6105].

This specification does not improve the Neighbor Discovery Protocol security model, but extends the purely link-local trust relationship between the host and the default routers with HTTP over TLS communications which servers are authenticated as rightful owners of the FQDN received within the trusted PvD ID RA option.

It must be noted that Section 4.4 of this document only provides reasonable assurance against misconfiguration but does not prevent an hostile network access provider to wrong information that could lead applications or hosts to select an hostile PvD. Users should always apply caution when connecting to an unknown network.

7. Privacy Considerations

Retrieval of the PvD Additional Information over HTTPS requires early communications between the connecting host and a server which may be located further than the first hop router. Although this server is likely to be located within the same administrative domain as the default router, this property can't be ensured. Therefore, hosts willing to retrieve the PvD Additional Information before using it without leaking identity information, SHOULD make use of an IPv6 Privacy Address and SHOULD NOT include any privacy sensitive data, such as User Agent header or HTTP cookie, while performing the HTTP over TLS query.

From a privacy perspective, retrieving the PvD Additional Information is not different from establishing a first connexion to a remote server, or even performing a single DNS lookup. For example, most operating systems already perform early queries to well known web sites, such as <http://captive.example.com/hotspot-detect.html>, in order to detect the presence of a captive portal.

There may be some cases where hosts, for privacy reasons, should refrain from accessing servers that are located outside a certain network boundary. In practice, this could be implemented as a whitelist of 'trusted' FQDNs and/or IP prefixes that the host is allowed to communicate with. In such scenarios, the host SHOULD check that the provided PvD ID, as well as the IP address that it resolves into, are part of the allowed whitelist.

8. IANA Considerations

IANA is asked to assign the value TBD from the IPv6 Neighbor Discovery Option Formats registry for the PvD ID Router Advertisement option.

IANA is asked to assign the value "pvd" from the Well-Known URIs registry.

IANA is asked to create and maintain a new registry entitled "Additional Information PvD Keys" containing ASCII strings. The initial content of this registry are given in Section 4.3; future assignments are to be made through Expert Review [BCP36].

Finally, IANA is asked to create and maintain a new registry entitled "PvD ID Router Advertisement option Flags" reserving bit positions from 0 to 15 to be used in the PvD ID Router Advertisement option bitmask. Bit position 0, 1 and 2 are reserved by this document (as specified in Figure 1). Future assignments require a Standard Track RFC document.

9. Acknowledgements

Many thanks to M. Stenberg and S. Barth for their earlier work: [I-D.stenberg-mif-mpvd-dns], as well as to Basile Bruneau who was author of an early version of this document.

Thanks also to Marcus Keane, Mikael Abrahamson, Ray Bellis, Lorenzo Colitti, Bob Hinden, Tatuya Jinmei, Erik Kline, Ted Lemon, Jen Lenkova, Mark Townsley and James Woodyatt for useful and interesting discussions.

Finally, special thanks to Thierry Danis and Wenqin Shao for their valuable inputs and implementation efforts ([github]), Tom Jones for his integration effort into the NEAT project and Rigil Salim for his implementation work.

10. References

10.1. Normative references

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <<https://www.rfc-editor.org/info/rfc1035>>.

[RFC1982] Elz, R. and R. Bush, "Serial Number Arithmetic", RFC 1982, DOI 10.17487/RFC1982, August 1996, <<https://www.rfc-editor.org/info/rfc1982>>.

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC2461] Narten, T., Nordmark, E., and W. Simpson, "Neighbor Discovery for IP Version 6 (IPv6)", RFC 2461, DOI 10.17487/RFC2461, December 1998, <<https://www.rfc-editor.org/info/rfc2461>>.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <<https://www.rfc-editor.org/info/rfc2818>>.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <<https://www.rfc-editor.org/info/rfc3629>>.
- [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, <<https://www.rfc-editor.org/info/rfc4861>>.
- [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <<https://www.rfc-editor.org/info/rfc7159>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.

10.2. Informative references

- [github] Cisco, "IPv6-mPvD github repository", <<https://github.com/IPv6-mPvD>>.
- [I-D.kline-mif-mpvd-api-reqs] Kline, E., "Multiple Provisioning Domains API Requirements", draft-kline-mif-mpvd-api-reqs-00 (work in progress), November 2015.
- [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.

- [IEEE8021X] IEEE, "IEEE Standards for Local and Metropolitan Area Networks: Port based Network Access Control, IEEE Std".
- [PEN] IANA, "Private Enterprise Numbers",
<<https://www.iana.org/assignments/enterprise-numbers>>.
- [RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002,
<<https://www.rfc-editor.org/info/rfc3339>>.
- [RFC3971] Arkko, J., Ed., Kempf, J., Zill, B., and P. Nikander, "SEcure Neighbor Discovery (SEND)", RFC 3971, DOI 10.17487/RFC3971, March 2005,
<<https://www.rfc-editor.org/info/rfc3971>>.
- [RFC4191] Draves, R. and D. Thaler, "Default Router Preferences and More-Specific Routes", RFC 4191, DOI 10.17487/RFC4191, November 2005, <<https://www.rfc-editor.org/info/rfc4191>>.
- [RFC4941] Narten, T., Draves, R., and S. Krishnan, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 4941, DOI 10.17487/RFC4941, September 2007,
<<https://www.rfc-editor.org/info/rfc4941>>.
- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010,
<<https://www.rfc-editor.org/info/rfc5785>>.
- [RFC6105] Levy-Abegnoli, E., Van de Velde, G., Popoviciu, C., and J. Mohacsi, "IPv6 Router Advertisement Guard", RFC 6105, DOI 10.17487/RFC6105, February 2011,
<<https://www.rfc-editor.org/info/rfc6105>>.
- [RFC6296] Wasserman, M. and F. Baker, "IPv6-to-IPv6 Network Prefix Translation", RFC 6296, DOI 10.17487/RFC6296, June 2011,
<<https://www.rfc-editor.org/info/rfc6296>>.
- [RFC6724] Thaler, D., Ed., Draves, R., Matsumoto, A., and T. Chown, "Default Address Selection for Internet Protocol Version 6 (IPv6)", RFC 6724, DOI 10.17487/RFC6724, September 2012,
<<https://www.rfc-editor.org/info/rfc6724>>.
- [RFC7556] Anipko, D., Ed., "Multiple Provisioning Domain Architecture", RFC 7556, DOI 10.17487/RFC7556, June 2015,
<<https://www.rfc-editor.org/info/rfc7556>>.

- [RFC8028] Baker, F. and B. Carpenter, "First-Hop Router Selection by Hosts in a Multi-Prefix Network", RFC 8028, DOI 10.17487/RFC8028, November 2016, <<https://www.rfc-editor.org/info/rfc8028>>.
- [RFC8106] Jeong, J., Park, S., Beloeil, L., and S. Madanapalli, "IPv6 Router Advertisement Options for DNS Configuration", RFC 8106, DOI 10.17487/RFC8106, March 2017, <<https://www.rfc-editor.org/info/rfc8106>>.

Appendix A. Changelog

Note to RFC Editors: Remove this section before publication.

A.1. Version 00

Initial version of the draft. Edited by Basile Bruneau + Eric Vyncke and based on Basile's work.

A.2. Version 01

Major rewrite intended to focus on the the retained solution based on corridors, online, and WG discussions. Edited by Pierre Pfister. The following list only includes major changes.

PvD ID is an FQDN retrieved using a single RA option. This option contains a sequence number for push-based updates, a new H-flag, and a L-flag in order to link the PvD with the IPv4 DHCP server.

A lifetime is included in the PvD ID option.

Detailed Hosts and Routers specifications.

Additional Information is retrieved using HTTP-over-TLS when the PvD ID Option H-flag is set. Retrieving the object is optional.

The PvD Additional Information object includes a validity date.

DNS-based approach is removed as well as the DNS-based encoding of the PvD Additional Information.

Major cut in the list of proposed JSON keys. This document may be extended later if need be.

Monetary discussion is moved to the appendix.

Clarification about the 'prefixes' contained in the additional information.

Clarification about the processing of DHCPv6.

A.3. Version 02

The FQDN is now encoded with ASCII format (instead of DNS binary) in the RA option.

The PvD ID option lifetime is removed from the object.

Use well known URI "https://<PvD-ID>/.well-known/pvd"

Reference RFC3339 for JSON timestamp format.

The PvD ID Sequence field has been extended to 16 bits.

Modified host behavior for DHCPv4 and DHCPv6.

Removed IKEv2 section.

Removed mention of RFC7710 Captive Portal option. A new I.D. will be proposed to address the captive portal use case.

A.4. WG Document version 00

Document has been accepted as INTAREA working group document

IANA considerations follow RFC8126 [RFC8126]

PvD ID FQDN is encoded as per RFC 1035 [RFC1035]

PvD ID FQDN is prepended by a one-byte length field

Marcus Keane added as co-author

dnsZones key is added back

draft of a privacy consideration section and added that a temporary address should be used to retrieve the PvD additional information

per Bob Hinden's request: the document is now aiming at standard track and security considerations have been moved to the main section

A.5. WG Document version 01

Removing references to 'metered' and 'characteristics' keys. Those may be in scope of the PvD work, but this document will focus on essential parts only.

Removing appendix section regarding link quality and billing information.

The PvD RA Option may now contain other RA options such that PvD-aware hosts may receive configuration information otherwise invisible to non-PvD-aware hosts.

Clarify that the additional PvD Additional Information is not intended to modify host's networking stack behavior, but rather provide information to the Application, used to select which PvDs must be used and provide configuration parameters to the transport layer.

The RA option padding is used to increase the option size to the next 64 (was 32) bits boundary.

Better detail the Security model and Privacy considerations.

Authors' Addresses

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

Email: ppfister@cisco.com

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

Email: evyncke@cisco.com

Tommy Pauly
Apple

Email: tpauly@apple.com

David Schinazi
Apple

Email: dschinazi@apple.com

Internet Area WG
Internet Draft
Intended status: Best Current Practice
Updates: 4459
Expires: July 2018

J. Touch
M. Townsley
Cisco
January 19, 2018

IP Tunnels in the Internet Architecture
draft-ietf-intarea-tunnels-08.txt

Status of this Memo

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

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/ietf/lid-abstracts.txt>

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

This Internet-Draft will expire on July 19, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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.....	3
2. Conventions used in this document.....	6
2.1. Key Words.....	6
2.2. Terminology.....	6
3. The Tunnel Model.....	10
3.1. What is a Tunnel?.....	11
3.2. View from the Outside.....	13
3.3. View from the Inside.....	14
3.4. Location of the Ingress and Egress.....	15
3.5. Implications of This Model.....	15
3.6. Fragmentation.....	16
3.6.1. Outer Fragmentation.....	16
3.6.2. Inner Fragmentation.....	18
3.6.3. The Necessity of Outer Fragmentation.....	19
4. IP Tunnel Requirements.....	20
4.1. Encapsulation Header Issues.....	20
4.1.1. General Principles of Header Fields Relationships...20	
4.1.2. Addressing Fields.....	21
4.1.3. Hop Count Fields.....	21

4.1.4. IP Fragment Identification Fields.....	22
4.1.5. Checksums.....	23
4.2. MTU Issues.....	24
4.2.1. Minimum MTU Considerations.....	24
4.2.2. Fragmentation.....	27
4.2.3. Path MTU Discovery.....	30
4.3. Coordination Issues.....	32
4.3.1. Signaling.....	32
4.3.2. Congestion.....	34
4.3.3. Multipoint Tunnels and Multicast.....	34
4.3.4. Load Balancing.....	35
4.3.5. Recursive Tunnels.....	36
5. Observations.....	37
5.1. Summary of Recommendations.....	37
5.2. Impact on Existing Encapsulation Protocols.....	37
5.3. Tunnel Protocol Designers.....	40
5.3.1. For Future Standards.....	40
5.3.2. Diagnostics.....	40
5.4. Tunnel Implementers.....	41
5.5. Tunnel Operators.....	41
6. Security Considerations.....	42
7. IANA Considerations.....	43
8. References.....	43
8.1. Normative References.....	43
8.2. Informative References.....	43
9. Acknowledgments.....	49
APPENDIX A: Fragmentation efficiency.....	50
A.1. Selecting fragment sizes.....	50
A.2. Packing.....	51

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).

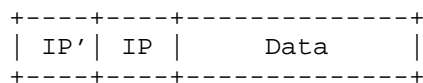


Figure 1 IP inside IP

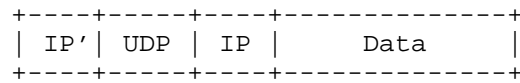


Figure 2 IP in UDP in IP in Ethernet

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:

- o IP in IP / mobile IP - IPv4 in IPv4 tunnels using protocol 4 [RFC2003][RFC2473][RFC5944] and its precursor called "IPIP" using protocol 94 [RFC1853]
- o IP in IPv6 - IPv6 or IPv4 in IPv6 [RFC2473]
- o IPsec - includes a tunnel mode to enable encryption or authentication of the an entire IP datagram inside another IP datagram [RFC4301]
- o 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]
- o 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
- o 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]

- o 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]
- o Generic UDP Encapsulation (GUE) - IP in UDP in IP [He17]
- o Automatic Multicast Tunneling (AMT) - IP in UDP in IP for multicast [RFC7450]
- o L2TP - PPP over IP, to extend a subscriber's DSL/FTTH connection from an access line provider to an ISP [RFC3931]
- o 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
- o L3VPNs - provides a network topology different from that provided by ISPs [RFC4176]
- o NVO3 - data center network sharing (to be determined, which may include use of GUE or other tunnels) [RFC7364]
- o PWE3 - emulates wire-like services over packet-switched services [RFC3985]
- o SEAL/AERO -IP in IP tunneling with an additional shim header designed to overcome the limitations of RFC2003 [RFC5320][Te18]
- o A number of legacy variants, including swIPe (an IPsec precursor), a GRE precursor, and the Internet Encapsulation Protocol, all of which included a shim layer [RFC1853]

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 size (i.e., Maximum Transmission Unit or MTU) mismatches 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:

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

- o 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.
- o 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].
- o Source or sender: the node that generates a message [RFC1122].
- o Destination or receiver: the node that consumes a message [RFC1122].
- o 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 [Tol6].
- o 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].
- o Link packet: a link layer message, which can carry an IP datagram as a payload
- o (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.
- o Path: a sequence of one or more links over which an IP message traverses between source and destination nodes (hosts or routers).
- o (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 [RFC8200].

- o 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 possible, EMTU_S = EMTU_R. 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 [RFC8200].
- o 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 [RFC8200].
- o Path MTU (PMTU): the largest message that can transit a path of links [RFC1191][RFC8201]. 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.
- o 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.
- o 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".

- o 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.
- o 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.
- o 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.
- o Inner header: the header of the message as it arrives to the ingress [RFC2003].
- o Outer header(s): one or more headers added to the message by the ingress, as part of the encapsulation for tunnel transit [RFC2003].
- o Mid-tunnel fragmentation: Fragmentation of the message during the tunnel transit, as could occur for IPv4 datagrams with DF=0 [RFC2983].
- o Atomic packet, datagram, or fragment: an IP packet that has not been fragmented and which cannot be fragmented further [RFC6864] [RFC6946].

The following terms are introduced by this document:

- o (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.

- o (Tunnel) link packet (TLP): packets that traverse between two interfaces, e.g., from ingress interface to egress interface, in which resides all or part of a transit packet. A tunnel link packet is the tunnel equivalent of a link (layer) packet as it would traverse a link, which is why we use the same terminology.
- o 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.
- o Tunnel maximum atomic packet (MAP): 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.
- o 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.
- o 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.
- o 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.
- o EMFS_S: the link layer equivalent of EMTU_S.
- o EMFS_R: the link layer equivalent of EMTU_R.
- o 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:

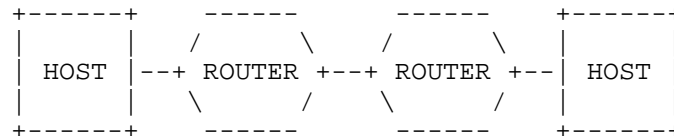


Figure 3 Basic Internet architecture

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.

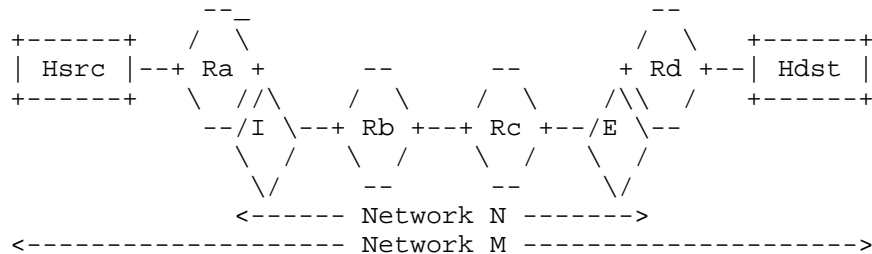


Figure 4 The big picture

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.

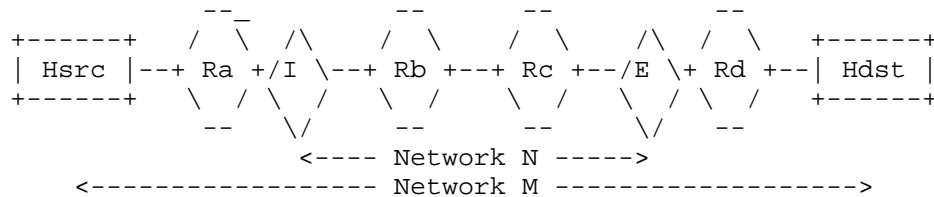


Figure 5 IP in IP network picture

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].

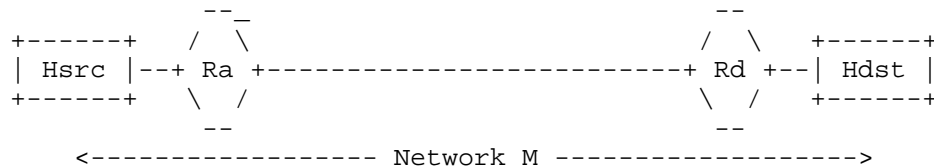


Figure 6 Tunnels as viewed from the outside

For example, the IP datagram hop counts (IPv4 Time-to-Live [RFC791] and IPv6 Hop Limit [RFC8200]) 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. This 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).

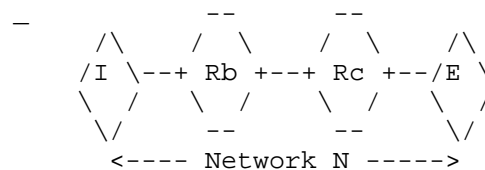


Figure 7 Tunnels, as viewed from within the tunnel

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.

An ingress or egress can be implemented as an integrated component, appearing equivalent to any other network interface, or can be more complex. In the simple variant, each is tightly coupled to another network interface, e.g., where the ingress emits encapsulated packets directly into another network interface, or where the egress receives packets to decapsulate directly from another network interface.

The other implementation variant is more modular, but more complex to explain. The ingress acts like a network interface by receiving IP packets to transmit from an upper layer protocol (or relay mechanism of a router), but then acts like an upper layer protocol (or relay mechanism of a router) when it emits encapsulated packets back into the same node. The egress acts like an upper layer interface (or relay mechanism of a router) by receiving packets from a network interface, but then acts like a network interface when it emits decapsulated packets back in to the same node. To the existing network interfaces, the ingress/egress act like upper layer interfaces (i.e., sending or receiving application stacks), while to the interior of the node, the ingress/egress act like network interfaces. This dual nature inside the node reflects the duality of the tunnel as transit link and host-host channel.

3.5. Implications of This Model

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

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

- o Like a link MTU, a tunnel MTU is defined by the effective MTU of the receiver (i.e., EMTU_R of the egress).
- o 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.
- o The tunnel path MFS is not relevant to the transited traffic. There is no mechanism or protocol by which it can be determined.
- o 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.
- o 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.
- o 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.
- o 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.

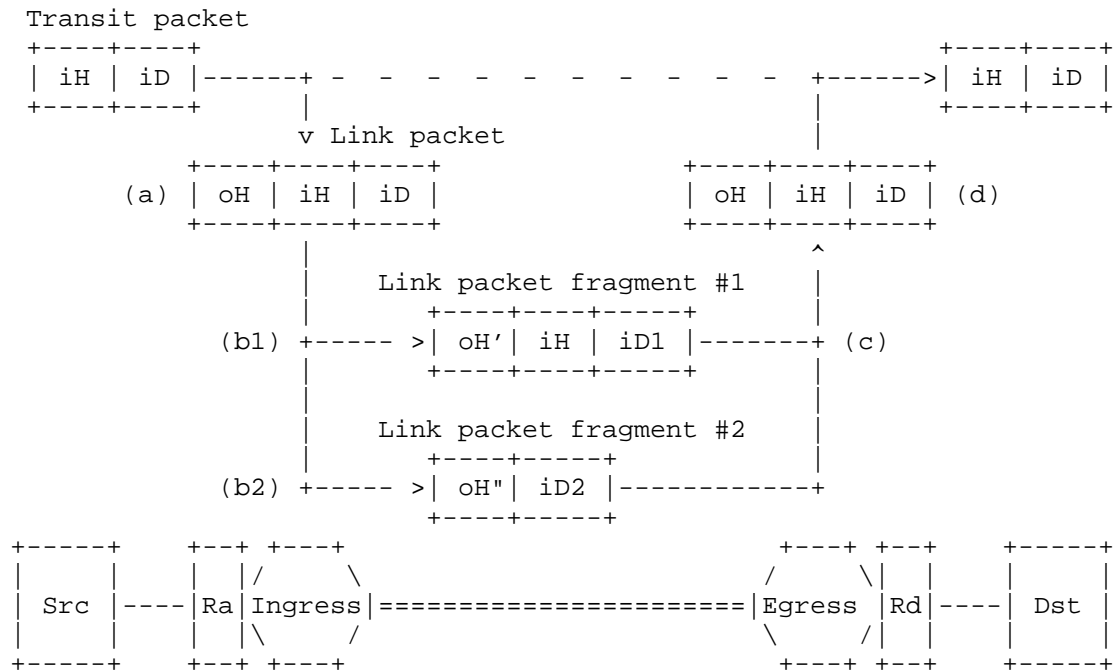


Figure 8 Fragmentation of the (outer) link packet

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.

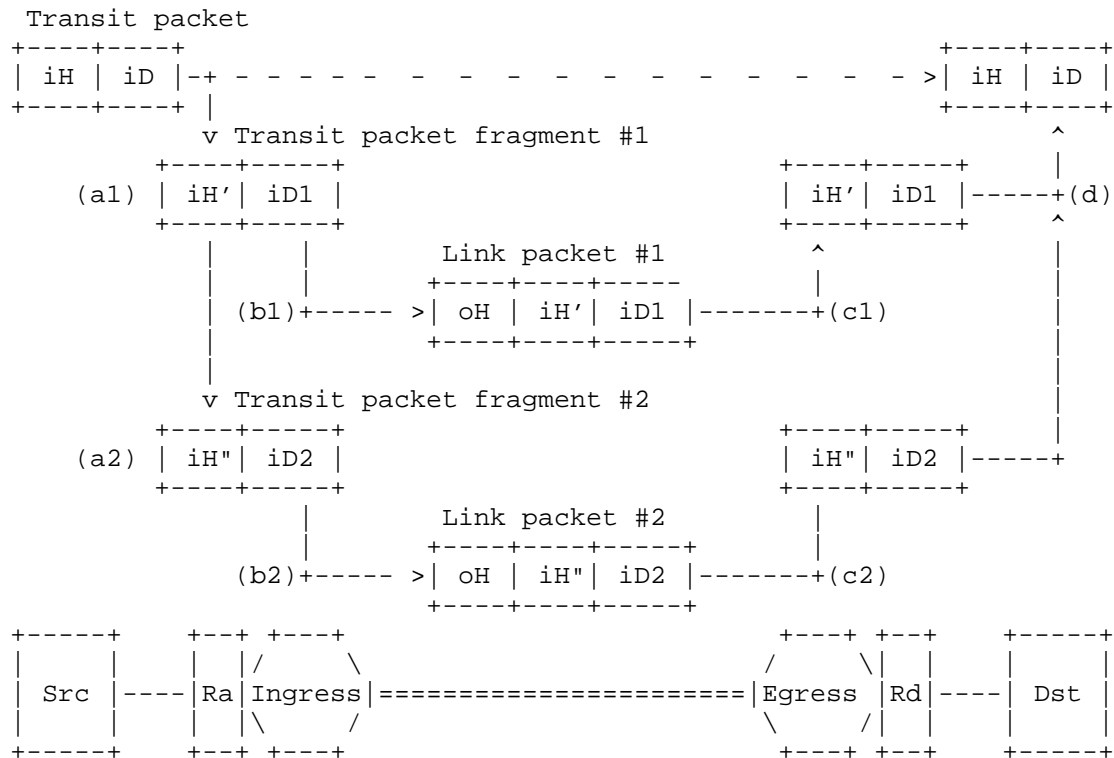


Figure 9 Fragmentation of the inner (transit) packet

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 transmits 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 [RFC8200] 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

Tunnel 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][RFC8200]. 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][RFC8200]. 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 [RFC8200]. 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 [RFC6864][RFC8200].

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, 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 [RFC6864]. 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 [RFC8200]. 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][Tel8] and GRE [RFC2784] as well as legacy protocols swIpe and the Internet Encapsulation Protocol [RFC1853], 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][RFC8200] and by a variety of protocol mechanisms that attempt to establish relationships between them, including path MTU discovery (PMTUD) [RFC1191][RFC8201], 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.

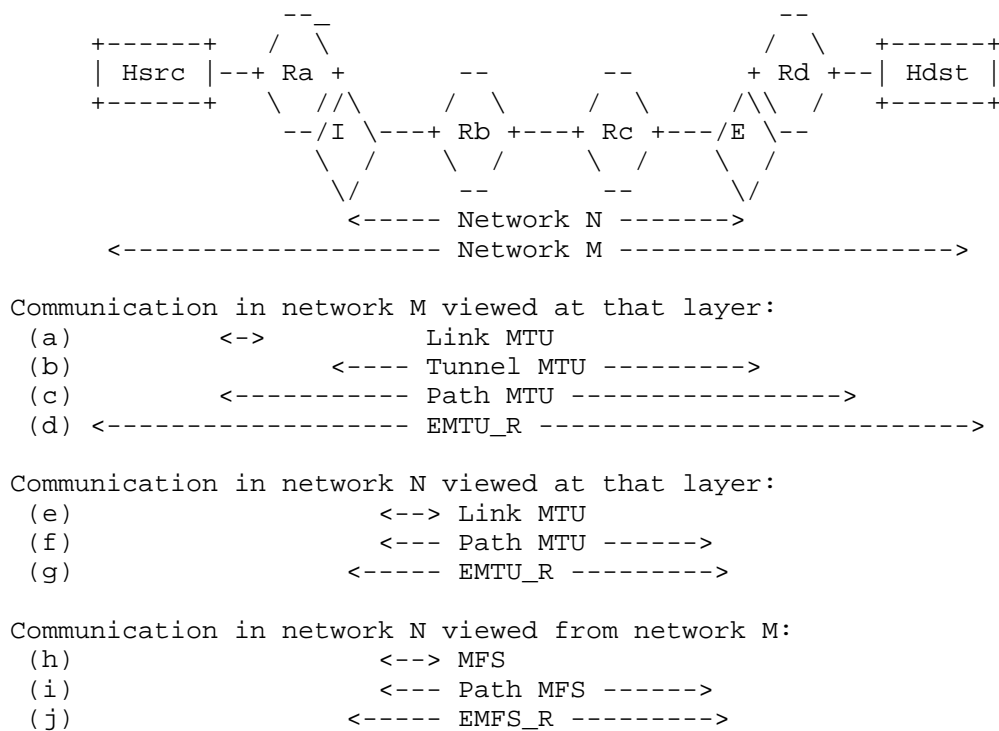


Figure 10 The variety of MTU values

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][RFC8201]) 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 (d) 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 message maximum is limited by the path MTU, and the source-destination message maximum (EMTU_S) is limited

by the path MTU when source fragmentation is disabled and by EMTU_R otherwise, 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 (h)). 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 (j). The tunnel EMTU_R is EMFS_R minus the link (encapsulation) headers and includes the encapsulation headers of the link layer. Just as the path MTU has no bearing on EMTU_R, the path MFS (i) in network N has no bearing on the MTU of the tunnel.

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

- o Network M MTU = 1280, the largest transit packet (i.e., payload) over a single IPv6 link in the base network without source fragmentation
- o 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
- o Network M EMTU_R = 1500, the largest transit packet (i.e., payload) that can traverse a path in the base network with source fragmentation
- o Network N MTU = 1280 (for the same reasons as for network M)
- o Network N path MTU = 1280 (for the same reasons as for network M)
- o Network N EMTU_R = 1500 (for the same reasons as for network M)
- o Tunnel MTU = 1500-encapsulation (typically 1460), the network N EMTU_R payload
- o Tunnel MAP (maximum atomic packet) = largest network M message that transits a tunnel as an atomic packet using network N as a link layer: 1280-encapsulation, i.e., the network N path MTU payload (which is itself limited by the tunnel path MFS)

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 [RFC8200]. 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:

- o TP: transit packet
- o TLP: tunnel link packet
- o TPsize: size of the transit packet (including its headers)
- o encaps: ingress encapsulation overhead (tunnel link headers)
- o tunMTU: tunnel MTU, i.e., network N egress EMTU_R - encaps
- o tunMAP: tunnel maximum atomic packet as limited by the tunnel path MFS

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 TP fragments of tunMTU size
    and send each TP fragment to the tunnel ingress interface
  else
    drop the TP and send ICMP "too big" to the TP source
  endif
else
  send TP to the tunnel ingress (i.e., as an outbound interface)
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 <= tunMAP) then
  encapsulate the TP and emit
else
  if (tunMAP < TPsize) then
    encapsulate the TP, creating the TLP
    fragment the TLP into tunMAP chunks
    emit the TLP fragments
  endif
endif
```

Figure 12 Ingress processing algorithm

Note that these Figure 11 and Figure 12 indicate that a node might both "fragment then encapsulate" and "encapsulate then fragment", i.e., the effect is "on-path fragment, then encapsulate, then source fragment". The first (on-path) fragmentation occurs only for IPv4 DF=0 packets, based on the tunnel MTU. The second (source) fragmentation occurs for all packets, based on the tunnel maximum atomic packet (MAP) size. The first fragmentation is a convenience for a subset of IPv4 packets; it is the second (source) fragmentation that ensures that messages traverse the tunnel.

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 $1500 - (40 + \text{EHsize})$ bytes. The tunnel path MTU will be at least $1280 - (40 + \text{EHsize})$ bytes, which then also represents the tunnel maximum atomic packet size (MAP). Transit packets larger than the tunnel MTU will be dropped by a node before ingress processing, and so do not need to be addressed as part of 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
        encapsulate the TP, creating the TLP
        fragment the TLP into (1240 - EHsize) chunks
        emit the TLP fragments
    endif
endif
```

Figure 13 Ingress processing for an tunnel over IPv6

IPv6 cannot necessarily support all tunnel encapsulations. When the egress EMTU_R is the default of 1500 bytes, 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. Under the same EMTU_R assumption, 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 [Tel8]). In particular, many tunnel specifications are often able to avoid persistent fragmentation because they operationally assume larger EMTU_R and tunnel MAP sizes than are guaranteed for IPv4 [RFC1122] or IPv6 [RFC8200].

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 MAP size rather than to 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 big 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:

- o 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.
- o 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".
- o 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 "packet to big" ICMP packets in the reverse direction of the tunnel [RFC1191][RFC8201]. 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].

PLPMTUD might require that the ingress consider the potential impact of multipath forwarding (see Section 4.3.4). In such cases, probes generated by the ingress might need to track different flows, e.g., that might traverse different tunnel paths. Additionally, encapsulation might need to consider mechanisms to ensure that probes traverse the same path as their corresponding traffic, even when labeled as the same flow (e.g., using the IPv6 flow ID). In such cases, the transit packet and probe may need to be encrypted or encapsulated in an additional flow-based transport header, to avoid differential path traversal based on deep-packet inspection within the tunnel.

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).

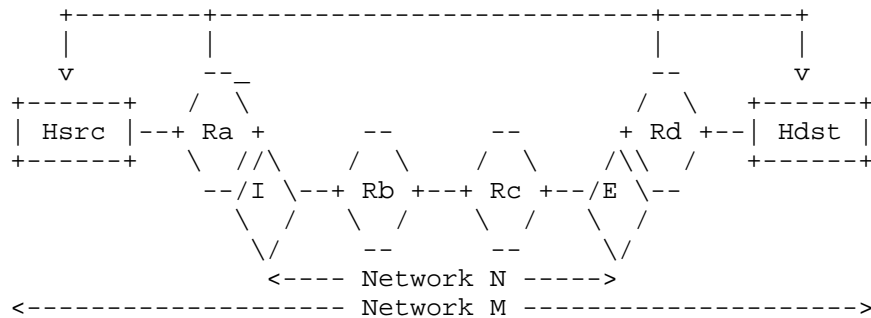


Figure 14 Signals outside the tunnel

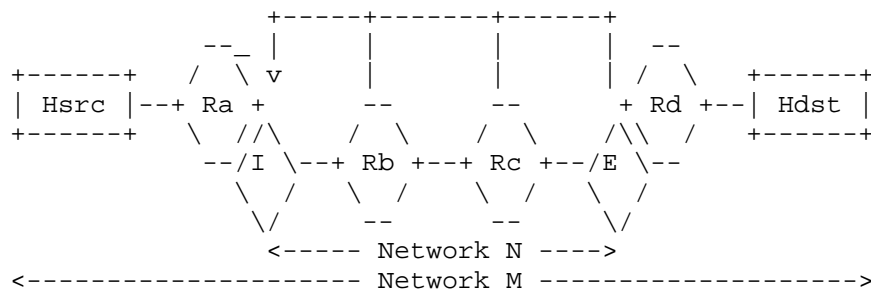


Figure 15 Signals inside the tunnel

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 atomic packet or packet fragment size. 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_R 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 [RFC2529][RFC5214][Tel8]. 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 egress EMTU_R (the latter determining the tunnel MTU). Values valid across all receivers need to be confirmed in advance (e.g., via IPv6 ND announcements or out-of-band configuration information) before a multipoint tunnel or link can use values other than the default, otherwise packets may reach some receivers but be "black-holed" to others (e.g., if PMTUD fails [RFC2923]).

A multipoint tunnel MUST have support for broadcast and multicast (or their equivalent), 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., NBMA).

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 tunnel path MTU or tunnel MAP values, causing the ingress

EMTU_S to vary [RFC7690]. When individual values cannot be correlated to transit traffic, the EMTU_S can be set to the minimum of these different path MTU and MAP values.

In some cases, these values can be correlated to paths, e.g., IPv6 packets include a flow label to enable multipath routing to keep packets of a single flow following the same path, as well as to help differentiate path properties (e.g., for path MTU discovery [RFC4821]). It is important to preserve the semantics of that flow label as an aggregate identifier of 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 appropriate 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.

When the transit packet is visible to forwarding nodes inside the tunnel (e.g., when it is not encrypted), those nodes use deep packet inspection (DPI) context to send a single flow over different paths. This sort of "DPI override" of the IP flow information can interfere with both PMTUD and PLPMTUD mechanisms. The only way to ensure that intermediate nodes do not interfere with PLPMTUD is to encrypt the transit packet when it is encapsulated for tunnel traversal, or to provide some other signals (e.g., an additional layer of encapsulation header including transport ports) that preserves the flow semantics.

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 [He17]).

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

- o 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 do 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_R less headers, and not related at all to the ingress-egress MFS
- o Tunnels must obey core IP requirements
 - o Obey IPv4 DF=1 on arrival at a node (nodes MUST NOT fragment IPv4 packets where DF=1 and routers MUST NOT clear the DF bit)
 - 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]
 - o Treats tunnel MTU as tunnel path MTU, not tunnel egress MTU
 - o Requires ingress to generate ICMP errors
 - o Copies inner hop limit to outer
- o L2TP [RFC3931]
 - o Treats tunnel MTU as tunnel path MTU, not tunnel egress MTU
 - o Requires ingress to generate ICMP errors
- o PWE [RFC3985]
 - o Treats tunnel MTU as tunnel path MTU, not tunnel egress MTU
 - o Requires ingress to generate ICMP errors
- o GUE (Generic UDP encapsulation) [He17] - IP (et. al) in UDP in IP
 - o Allows inner encapsulation fragmentation
- o Geneve [RFC7364][Gr18] - IP (et al.) in Geneve in UDP in IP
 - o Treats tunnel MTU as tunnel path MTU, not tunnel egress MTU
- o SEAL/AERO [RFC5320][Te18] - IP in SEAL/AERO in IP
 - o Some issues with SEAL (MTU, ICMP), corrected in AERO
- o RTG DT encapsulations [No16]
 - o Assumes fragmentation can be avoided completely
 - o Allows encapsulation protocols that lack fragmentation
 - o Relies on ICMP PTB to correct for tunnel path MTU
- o No known issues

- o L2VPN (framework for L2 virtualization) [RFC4664]
- o L3VPN (framework for L3 virtualization) [RFC4176]
- o MPLS (IP in MPLS) [RFC3031]
- o 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 Gorrry'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.

[Gr18] Gross, J. (Ed.), I. Ganga (Ed.), T. Sridhar (Ed.), "Geneve: Generic Network Virtualization Encapsulation," draft-ietf-nvo3-geneve-05, Sep. 2017.

[Hel17] Herbert, T., L. Yong, O. Zia, "Generic UDP Encapsulation," draft-ietf-intarea-gue-05, Dec. 2017.

[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.
- [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.
- [RFC2473] Conta, A., "Generic Packet Tunneling in IPv6 Specification," RFC 2473, Dec. 1998.
- [RFC2529] Carpenter, B., C. Jung, "Transmission of IPv6 over IPv4 Domains without Explicit Tunnels," RFC 2529, Mar. 1999.
- [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.
- [RFC5214] Templin, F., T. Gleeson, D. Thaler, "Intra-Site Automatic Tunnel Addressing Protocol (ISATAP)," RFC 5214, Mar. 2008.
- [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.
- [RFC8200] Deering, S., R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," RFC 8200, Jul. 2017.
- [RFC8201] McCann, J., S. Deering, J. Mogul, R. Hinden (Ed.), "Path MTU Discovery for IP version 6," RFC 8201, Jul. 2017.
- [Sa84] Saltzer, J., D. Reed, D. Clark, "End-to-end arguments in system design," ACM Trans. on Computing Systems, Nov. 1984.
- [Tel8] Templin, F., "Asymmetric Extended Route Optimization," draft-templin-aerolink-78, Jan. 2018.
- [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, Gorrry 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
Manhattan Beach, CA 90266
U.S.A.

Phone: +1 (310) 560-0334
Email: touch@strayalpha.com

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.

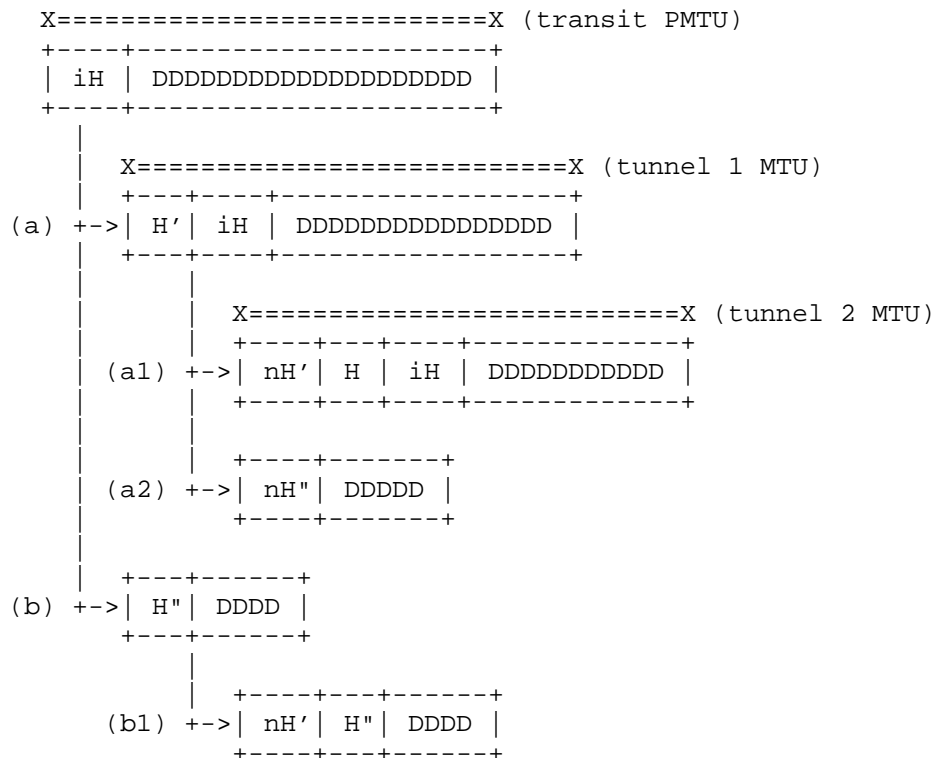


Figure 16 Fragmenting via maximum fit

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.

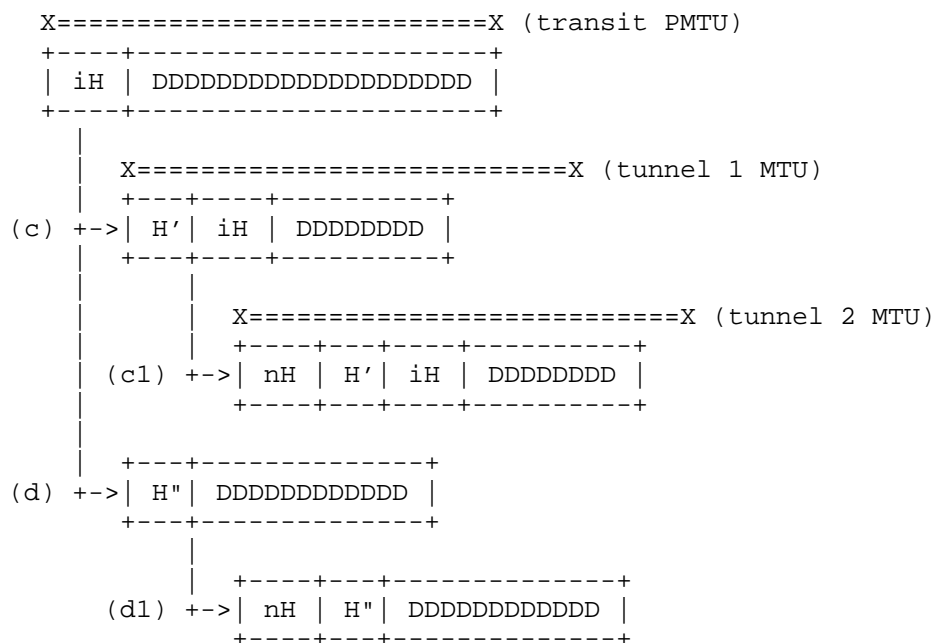


Figure 17 Fragmenting via "even split"

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.

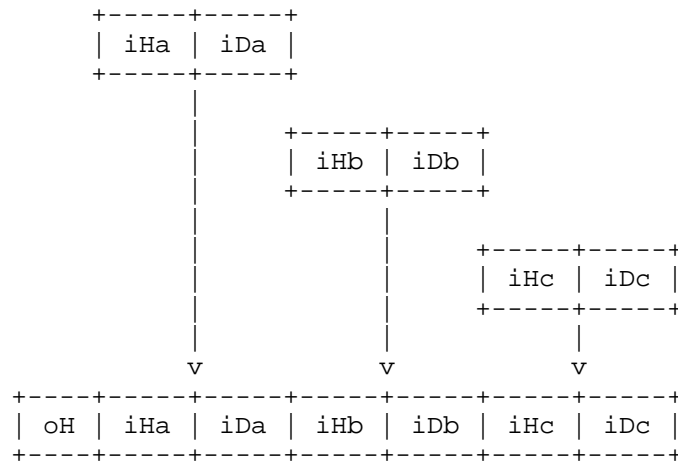


Figure 18 Packing packets into a tunnel

Internet Area Working Group
Internet-Draft
Intended status: Experimental
Expires: September 6, 2018

V. Olteanu
D. Niculescu
University Politehnica of Bucharest
March 05, 2018

SOCKS Protocol Version 6
draft-olteanu-intarea-socks-6-02

Abstract

The SOCKS protocol is used primarily to proxy TCP connections to arbitrary destinations via the use of a proxy server. Under the latest version of the protocol (version 5), it takes 2 RTTs (or 3, if authentication is used) before data can flow between the client and the server.

This memo proposes SOCKS version 6, which reduces the number of RTTs used, takes full advantage of TCP Fast Open, and adds support for 0-RTT authentication.

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 <https://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 6, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction	2
1.1. Revision log	3
2. Requirements language	5
3. Mode of operation	5
4. Connection Requests	6
5. Version Mismatch Replies	7
6. Authentication Replies	8
7. Operation Replies	9
7.1. Handling CONNECT	10
7.2. Handling BIND	11
7.3. Handling UDP ASSOCIATE	11
8. SOCKS Options	11
8.1. Socket options	11
8.1.1. TFO options	12
8.1.2. Multipath TCP options	13
8.1.3. MPTCP Scheduler options	14
8.2. Authentication Method options	14
8.3. Authentication Data options	15
8.4. Idempotence options	16
8.4.1. Requesting a fresh token window	17
8.4.2. Spending a token	18
8.4.3. Handling Token Window Advertisements	19
8.5. Salt options	19
9. Username/Password Authentication	20
10. Security Considerations	20
10.1. Large requests	20
10.2. Replay attacks	21
10.3. Identical request profiling	21
11. IANA Considerations	21
12. Acknowledgements	22
13. References	22
13.1. Normative References	22
13.2. Informative References	22
Authors' Addresses	22

1. Introduction

Versions 4 and 5 [RFC1928] of the SOCKS protocol were developed two decades ago and are in widespread use for circuit level gateways or as circumvention tools, and enjoy wide support and usage from various software, such as web browsers, SSH clients, and proxifiers.

However, their design needs an update in order to take advantage of the new features of transport protocols, such as TCP Fast Open [RFC7413], or to better assist newer transport protocols, such as MPTCP [RFC6824].

One of the main issues faced by SOCKS version 5 is that, when taking into account the TCP handshake, method negotiation, authentication, connection request and grant, it may take up to 5 RTTs for a data exchange to take place at the application layer. This is especially costly in networks with a large delay at the access layer, such as 3G, 4G, or satellite.

The desire to reduce the number of RTTs manifests itself in the design of newer security protocols. TLS version 1.3 [I-D.ietf-tls-tls13] defines a zero round trip (0-RTT) handshake mode for connections if the client and server had previously communicated.

TCP Fast Open [RFC7413] is a TCP option that allows TCP to send data in the SYN and receive a response in the first ACK, and aims at obtaining a data response in one RTT. The SOCKS protocol needs to concern itself with at least two TFO deployment scenarios: First, when TFO is available end-to-end (at the client, at the proxy, and at the server); second, when TFO is active between the client and the proxy, but not at the server.

This document describes the SOCKS protocol version 6. The key improvements over SOCKS version 5 are:

- o The client sends as much information upfront as possible, and does not wait for the authentication process to conclude before requesting the creation of a socket.
- o The connection request also mimics the semantics of TCP Fast Open [RFC7413]. As part of the connection request, the client can supply the potential payload for the initial SYN that is sent out to the server.
- o The protocol can be extended via options without breaking backward-compatibility.
- o The protocol can leverage the aforementioned options to support 0-RTT authentication schemes.

1.1. Revision log

Typos and minor clarifications are not listed.

draft-02

- o Made support for Idempotence options mandatory for proxies.
- o Clarified what happens when proxies can not or will not issue tokens.
- o Limited token windows to $2^{31} - 1$.
- o Fixed definition of "less than" for tokens.
- o NOOP commands now trigger Operation Replies.
- o Renamed Authentication options to Authentication Data options.
- o Authentication Data options are no longer mandatory.
- o Authentication methods are now advertised via options.
- o Shifted some Request fields.
- o Option range for vendor-specific options.
- o Socket options.
- o Password authentication.
- o Salt options.

draft-01

- o Added this section.
- o Support for idempotent commands.
- o Removed version numbers from operation replies.
- o Request port number for SOCKS over TLS. Deprecate encryption/encapsulation within SOCKS.
- o Added Version Mismatch Replies.
- o Renamed the AUTH command to NOOP.
- o Shifted some fields to make requests and operation replies easier to parse.

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. Mode of operation

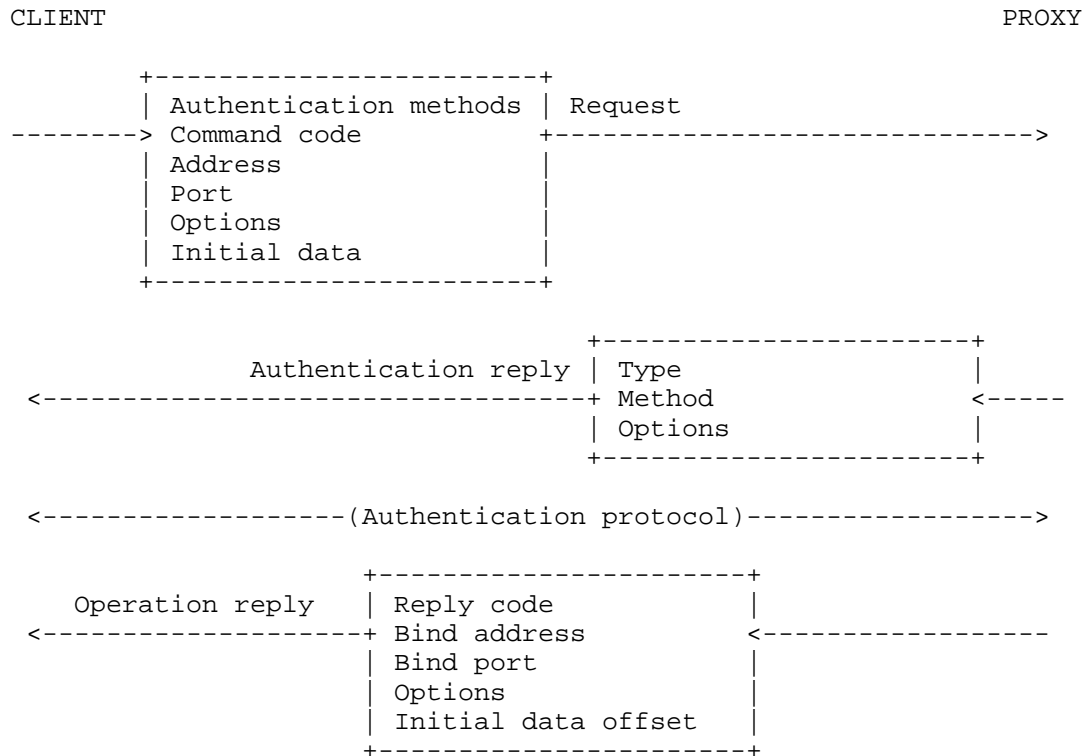


Figure 1: The SOCKS version 6 protocol message exchange

When a TCP-based client wishes to establish a connection to a server, it must open a TCP connection to the appropriate SOCKS port on the SOCKS proxy. The client then enters a negotiation phase, by sending the request in figure Figure 1, that contains, in addition to fields present in SOCKS 5 [RFC1928], fields that facilitate low RTT usage and faster authentication negotiation.

Next, the server sends an authentication reply. If the request did not contain the necessary authentication information, the proxy

indicates an authentication method that must proceed. This may trigger a longer authentication sequence that could include tokens for ulterior faster authentications. The part labeled "Authentication protocol" is specific to the authentication method employed and is not expected to be employed for every connection between a client and its proxy server. The authentication protocol typically takes up 1 RTT or more.

If the authentication is successful, an operation reply is generated by the proxy. It indicates whether the proxy was successful in creating the requested socket or not.

In the fast case, when authentication is properly set up, the proxy attempts to create the socket immediately after the receipt of the request, thus achieving an operational connection in one RTT (provided TFO functionality is available at the client, proxy, and server).

4. Connection Requests

The client starts by sending a request to the proxy.

Version		Command	Port	Address	Address
Major	Minor	Code		Type	
1	1	1	2	1	Variable
Number of Options		Options	Initial Data Size	Initial Data	
1		Variable	2	Variable	

Figure 2: SOCKS 6 Request

- o Version: The major byte MUST be set to 0x06, and the minor byte MUST be set to 0x00.
- o Command Code:
 - * 0x00 NOOP: authenticate the client and do nothing.
 - * 0x01 CONNECT: requests the establishment of a TCP connection.
 - * 0x02 BIND: requests the establishment of a TCP port binding.

- * 0x03 UDP ASSOCIATE: requests a UDP port association.
- o Address Type:
 - * 0x01: IPv4
 - * 0x03: Domain Name
 - * 0x04: IPv6
- o Address: this field's format depends on the address type:
 - * IPv4: a 4-byte IPv4 address
 - * Domain Name: one byte that contains the length of the FQDN, followed by the FQDN itself. The string is not NUL-terminated.
 - * IPv6: a 16-byte IPv6 address
- o Port: the port in network byte order.
- o Number of Options: the number of SOCKS options that appear in the Options field.
- o Options: see Section 8.
- o Initial Data Size: A two-byte number in network byte order. In case of NOOP, BIND or UDP ASSOCIATE, this field MUST be set to 0. In case of CONNECT, this is the number of bytes of initial data that are supplied in the following field.
- o Initial Data: The first octets of the data stream.

Clients can advertise their supported authentication methods by including an Authentication Method option (see Section 8.2).

The server MAY truncate the initial data to an arbitrary size and disregard the rest. This is will be communicated later to the client, should the authentication process be successful (see Section 7). As such, server implementations do not have to buffer the initial data while waiting for the (potentially malicious) client to authenticate.

5. Version Mismatch Replies

Upon receipt of a request starting with a version number other than 6.0, the proxy sends the following response:

Version	
Major	Minor
1	1

Figure 3: SOCKS 6 Version Mismatch Reply

- o Version: The major byte MUST be set to 0x06, and the minor byte MUST be set to 0x00.

A client MUST close the connection after receiving such a reply.

6. Authentication Replies

Upon receipt of a valid request, the proxy sends an Authentication Reply:

Version		Type	Method	Number of Options	Options
Major	Minor				
1	1	1	1	1	Variable

Figure 4: SOCKS 6 Authentication Reply

- o Version: The major byte MUST be set to 0x06, and the minor byte MUST be set to 0x00.
- o Type:
 - * 0x00: authentication successful.
 - * 0x01: further authentication needed.
- o Method: The chosen authentication method.
- o Number of Options: the number of SOCKS options that appear in the Options field.
- o Options: see Section 8.

Multihomed clients SHOULD cache the chosen method on a per-interface basis and SHOULD NOT include Authentication Data options related to

any other methods in further requests originating from the same interface.

If the server signals that further authentication is needed and selects "No Acceptable Methods", the client MUST close the connection.

The client and proxy begin a method-specific negotiation. During such negotiations, the proxy MAY supply information that allows the client to authenticate a future request using an Authentication Data option. The client and proxy SHOULD NOT negotiate the encryption of the application data. Descriptions of such negotiations are beyond the scope of this memo.

7. Operation Replies

After the authentication negotiations are complete, the server sends an Operation Reply:

Reply Code	Address Type	Bind Port	Bind Address	Initial Data Offset
1	1	2	Variable	2

Number of Options	Options
1	Variable

Figure 5: SOCKS 6 Operation Reply

o Reply Code:

- * 0x00: Success
- * 0x01: General SOCKS server failure
- * 0x02: Connection not allowed by ruleset
- * 0x03: Network unreachable
- * 0x04: Host unreachable
- * 0x05: Connection refused

- * 0x06: TTL expired
- * 0x07: Command not supported
- * 0x08: Address type not supported
- o Address Type:
 - * 0x01: IPv4
 - * 0x03: Domain Name
 - * 0x04: IPv6
- o Bind Address: the proxy bound address in the following format:
 - * IPv4: a 4-byte IPv4 address
 - * Domain Name: one byte that contains the length of the FQDN, followed by the FQDN itself. The string is not NUL-terminated.
 - * IPv6: a 16-byte IPv6 address
- o Bind Port: the proxy bound port in network byte order.
- o Number of Options: the number of SOCKS options that appear in the Options field.
- o Options: see Section 8.
- o Initial Data Offset: A two-byte number in network byte order. In case of BIND or UDP ASSOCIATE, this field MUST be set to 0. In case of CONNECT, it represents the offset in the plain data stream from which the client is expected to continue sending data.

If the proxy returns a reply code other than "Success", the client MUST close the connection.

If the client issued an NOOP command, the client MUST close the connection after receiving the Operation Reply.

7.1. Handling CONNECT

In case the client has issued a CONNECT request, data can now pass. The client MUST resume the data stream at the offset indicated by the Initial Data Offset field.

7.2. Handling BIND

In case the client has issued a BIND request, it must wait for a second Operation reply from the proxy, which signifies that a host has connected to the bound port. The Bind Address and Bind Port fields contain the address and port of the connecting host. Afterwards, application data may pass.

7.3. Handling UDP ASSOCIATE

The relay of UDP packets is handled exactly as in SOCKS 5 [RFC1928].

8. SOCKS Options

SOCKS options have the following format:

+-----+-----+		
Kind	Length	Option Data
+-----+-----+		
1	1	Variable
+-----+-----+		

Figure 6: SOCKS 6 Option

- o Kind: MUST be allocated by IANA. (See Section 11.)
- o Length: The length of the option.
- o Option Data: The contents are specific to each option kind.

8.1. Socket options

Socket options are be used by clients to alter the behavior of the sockets created by the proxy. A socket option can affect either the proxy's socket on the client-proxy leg or on the proxy-server leg. Clients can only place Socket options inside SOCKS Requests.

Proxies MAY include Socket options in their Operation Replies to signal their sockets' behavior. Said options MAY be unsolicited, i. e. the proxy MAY send them to signal behaviour that was not explicitly requested by the client.

Kind	Length	Leg	Level	Code	Data
1	1	2 bits	6 bits	1	Variable

Figure 7: Socket Option

- o Kind: MUST be allocated by IANA. (See Section 11.)
- o Length: The length of the option.
- o Leg:
 - * 0x1: Client-Proxy Leg
 - * 0x2: Proxy-Server Leg
 - * 0x3: Both Legs
- o Level:
 - * 0x01: Socket
 - * 0x02: IPv4
 - * 0x03: IPv6
 - * 0x04: TCP
 - * 0x05: UDP
- o Code: Option code
- o Data: Option-specific data

8.1.1. TFO options

Kind	Length	Leg	Level	Code
1	1	2 bits	6 bits	1

Figure 8: TFO Option

- o Kind: MUST be allocated by IANA. (See Section 11.)
- o Length: MUST be 4.
- o Leg: MUST be 0x2 (Proxy-Server Leg).
- o Level: 0x04 (TCP).
- o Code: 0x17

If a SOCKS Request contains a TFO option, the proxy SHOULD attempt to use TFO in case of a CONNECT command, or accept TFO in case of a BIND command. Otherwise, the proxy MUST NOT attempt to use TFO in case of a CONNECT command, or accept TFO in case of a BIND command.

In case of a CONNECT command, the proxy MAY include a TFO option in the Operation reply if TFO was attempted, the operation succeeded and the remote server supports TFO. In case of a BIND command, the proxy MAY include a TFO option in the first Operation reply to signal that it will accept an incoming TFO connection.

8.1.2. Multipath TCP options

In case of a CONNECT command, the proxy can inform the client that the connection to the server is an MPTCP connection.

Kind	Length	Leg	Level	Code
1	1	2 bits	6 bits	1

Figure 9: Multipath TCP Option

- o Kind: MUST be allocated by IANA. (See Section 11.)
- o Length: 4.
- o Leg: MUST be 0x2 (Proxy-Server Leg).
- o Level: 0x04 (TCP).
- o Code: 0x2a

8.1.3. MPTCP Scheduler options

In case of a CONNECT or BIND command, a client can use an MPTCP Scheduler option to indicate its preferred scheduler for the connection.

A proxy can use an MPTCP Scheduler option to inform the client about what scheduler is in use.

Kind	Length	Leg	Level	Code	Scheduler
1	1	2 bits	6 bits	1	1

Figure 10: MPTCP Scheduler Option

- o Kind: MUST be allocated by IANA. (See Section 11.)
- o Length: MUST be 5.
- o Leg: Either 0x01, 0x02, or 0x03 (Client-Proxy, Proxy-Client or Both legs).
- o Level: 0x04 (TCP).
- o Code: 0x2b
- o Scheduler:
 - * 0x00: Default
 - * 0x01: Round-Robin
 - * 0x02: Redundant

8.2. Authentication Method options

Authentication Method options are used by clients to advertise supported authentication methods. They can be part of SOCKS Requests.

Kind	Length	Methods
1	1	Variable

Figure 11: Authentication Method Option

- o Kind: MUST be allocated by IANA. (See Section 11.)
- o Length: The length of the option.
- o Methods: One byte per advertised method. Method numbers are assigned by IANA.

Clients MUST support the "No authentication required" method.
 Clients MAY omit advertising the "No authentication required" option.

8.3. Authentication Data options

Authentication Data options carry method-specific authentication data. They can be part of SOCKS Requests and Authentication Replies.

Authentication Data options have the following format:

Kind	Length	Method	Authentication Data
1	1	1	Variable

Figure 12: Authentication Data Option

- o Kind: MUST be allocated by IANA. (See Section 11.)
- o Length: The length of the option.
- o Method: The number of the authentication method. These numbers are assigned by IANA.
- o Authentication Data: The contents are specific to each method.

Clients MAY omit advertising authentication methods for which they have included at least an Authentication Data option.

8.4. Idempotence options

To protect against duplicate SOCKS Requests, authenticated clients can request, and then spend, idempotence tokens. A token can only be spent on a single SOCKS request.

Tokens are 4-byte unsigned integers in a modular 4-byte space. Therefore, if x and y are tokens, x is less than y if $0 < (y - x) < 2^{31}$ in unsigned 32-bit arithmetic.

Proxies grant contiguous ranges of tokens called token windows. Token windows are defined by their base (the first token in the range) and size. Windows can be shifted (i. e. have their base increased, while retaining their size) unilaterally by the proxy.

Requesting and spending tokens is done via Idempotence options:

Kind	Length	Type	Option Data
1	1	1	Variable

Figure 13: Idempotence Option

- o Kind: MUST be allocated by IANA. (See Section 11.)
- o Length: The length of the option.
- o Type:
 - * 0x00: Token Request
 - * 0x01: Token Window Advertisement
 - * 0x02: Token Expenditure
 - * 0x03: Token Expenditure Reply
- o Option Data: The contents are specific to each type.

All proxy implementations MUST support Idempotence options, even if they do not issue token windows.

8.4.1. Requesting a fresh token window

A client can obtain a fresh window of tokens by sending a Token Request option as part of a SOCKS Request:

Kind	Length	Type	Window Size
1	1	1	4

Figure 14: Token Request

- o Kind: MUST be allocated by IANA. (See Section 11.)
- o Length: 7
- o Type: 0x00 (Token Request)
- o Window Size: The requested window size.

If a token window is issued, the proxy then includes a Token Window Advertisement option in the corresponding Operation Reply:

Kind	Length	Type	Window Base	Window Size
1	1	1	4	4

Figure 15: Token Window Advertisement

- o Kind: MUST be allocated by IANA. (See Section 11.)
- o Length: 11
- o Type: 0x01 (Token Grant)
- o Window Base: The first token in the window.
- o Window Size: The window size. This value SHOULD be lower or equal to the requested window size. Window sizes MUST be less than 2^{31} .

If no token window is issued, the proxy MUST silently ignore the Token Request.

8.4.2. Spending a token

The client can attempt to spend a token by including a Token Expenditure option in its SOCKS request:

Kind	Length	Type	Token
1	1	1	4

Figure 16: Token Expenditure

- o Kind: MUST be allocated by IANA. (See Section 11.)
- o Length: 7
- o Type: 0x02 (Token Expenditure)
- o Token: The token being spent.

Clients SHOULD prioritize spending the smaller tokens.

The server responds by sending a Token Expenditure Reply option as part of the Operation Reply:

Kind	Length	Type	Response Code
1	1	1	1

Figure 17: Token Expenditure Response

- o Kind: MUST be allocated by IANA. (See Section 11.)
- o Length: 4
- o Type: 0x03 (Token Expenditure Response)
- o Response Code:
 - * 0x00: Success: The token was spent successfully.
 - * 0x01: No Window: The proxy does not have a token window associated with the client.

- * 0x02: Out of Window: The token is not within the window.
- * 0x03: Duplicate: The token has already been spent.

If eligible, the token is spent as soon as the client authenticates. If the token is not eligible for spending, the proxy **MUST NOT** attempt to honor the client's SOCKS Request; further, it **MUST** indicate a General SOCKS server failure in the Operation Reply.

Proxy implementations **SHOULD** also send a Token Window Advertisement if:

- o the token is out of window, or
- o by the proxy's internal logic, successfully spending the token caused the window to shift.

Proxy implementations **SHOULD NOT** shift the window's base beyond the highest unspent token.

Proxy implementations **MAY** include a Token Window Advertisement in any Operation Reply.

8.4.3. Handling Token Window Advertisements

Even though the proxy increases the window's base monotonically, there is no mechanism whereby a SOCKS client can receive the Token Window Advertisements in order. As such, clients **SHOULD** disregard unsolicited Token Window Advertisements with a Window Base less than the previously known value.

8.5. Salt options

Clients can use Salt options so that otherwise identical requests are unique. (See Section 10.3.)

+-----+-----+			
Kind	Length	Salt	
+-----+-----+			
1	1	4	
+-----+-----+			

Figure 18: Salt Option

- o Kind: **MUST** be allocated by IANA. (See Section 11.)
- o Length: 6

- o Salt: An arbitrary value.

Proxies MUST silently ignore Salt options.

9. Username/Password Authentication

Username/Password authentication is carried out as in [RFC1929].

Clients can also attempt to authenticate by placing the Username/Password request in an Authentication Data Option, provided that it is no longer than 252 bytes.

Kind	Length	Method	Username/Password request
1	1	1	Variable

Figure 19: Password authentication via a SOCKS Option

- o Kind: MUST be allocated by IANA. (See Section 11.)
- o Length: The length of the option.
- o Method: 0x02 (Username/Password).
- o Username/Password request: The Username/Password request, as described in [RFC1929].

10. Security Considerations

10.1. Large requests

Given the format of the request message, a malicious client could craft a request that is in excess of 100 KB and proxies could be prone to DDoS attacks.

To mitigate such attacks, proxy implementations SHOULD be able to incrementally parse the requests. Proxies MAY close the connection to the client if:

- o the request is not fully received after a certain timeout, or
- o the number of options exceeds an imposed hard cap, or
- o the total size of the options exceeds an imposed hard cap, or

- o the size of the initial data exceeds a hard cap.

Further, the server MAY choose not to buffer any initial data beyond what would be expected to fit in a TFO SYN's payload.

10.2. Replay attacks

In TLS 1.3, early data (which is likely to contain a full SOCKS request) is prone to replay attacks.

While Token Expenditure options can be used to mitigate replay attacks, the initial Token Request is still vulnerable. As such, client implementations SHOULD NOT make use of TLS early data when sending a Token Request.

10.3. Identical request profiling

If sent via TLS early data, identical SOCKS requests can also be identical on the wire. An attacker with the capability to capture a client's SOCKS traffic can attempt to profile it by identifying identical requests.

A client can use Salt options to make all of its requests unique.

11. IANA Considerations

This document requests that IANA allocate 1-byte option codes for SOCKS 6 options. Further, this document requests option codes for:

- o Socket options
- o Authentication Method options
- o Authentication Data options
- o Idempotence options
- o Salt options
- o Vendor-specific options

This document also requests that IANA allocate a port for SOCKS over TLS.

12. Acknowledgements

The protocol described in this draft builds upon and is a direct continuation of SOCKS 5 [RFC1928].

13. References

13.1. Normative References

[RFC1929] Leech, M., "Username/Password Authentication for SOCKS V5", RFC 1929, DOI 10.17487/RFC1929, March 1996, <<https://www.rfc-editor.org/info/rfc1929>>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

13.2. Informative References

[I-D.ietf-tls-tls13] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-26 (work in progress), March 2018.

[RFC1928] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and L. Jones, "SOCKS Protocol Version 5", RFC 1928, DOI 10.17487/RFC1928, March 1996, <<https://www.rfc-editor.org/info/rfc1928>>.

[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, <<https://www.rfc-editor.org/info/rfc6824>>.

[RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <<https://www.rfc-editor.org/info/rfc7413>>.

Authors' Addresses

Vladimir Olteanu
University Politehnica of Bucharest

Email: vladimir.olteanu@cs.pub.ro

Dragos Niculescu
University Politehnica of Bucharest

Email: dragos.niculescu@cs.pub.ro