Network Working Group
Internet-Draft
Intended status: Informational
Expires: April 17, 2019

B. Carpenter
Univ. of Auckland
B. Liu
Huawei Technologies

October 14, 2018

Limited Domains and Internet Protocols
draft-carpenter-limited-domains-04

Abstract

There is a noticeable trend towards network requirements, behaviours and semantics that are specific to a limited region of the Internet and a particular set of requirements. Policies, default parameters, the options supported, the style of network management and security requirements may vary. This document reviews examples of such limited domains and emerging solutions, and develops a related taxonomy. It then briefly discusses the standardization of protocols for limited domains. Finally, it shows the needs for a precise definition of limited domain membership and for mechanisms to allow nodes to join a domain securely and to find other members, including boundary nodes.

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 April 17, 2019.

Table of Contents

1. Introduction

As the Internet continues to grow and diversify, with a realistic prospect of tens of billions of nodes being connected directly and indirectly, there is a noticeable trend towards local requirements, behaviours and semantics. The word "local" should be understood in a special sense, however. In some cases it may refer to geographical and physical locality - all the nodes in a single building, on a single campus, or in a given vehicle. In other cases it may refer to a defined set of users or nodes distributed over a much wider area, but drawn together by a single virtual network over the Internet, or a single physical network running partially in parallel with the Internet. We expand on these possibilities below. To capture the topic, this document refers to such networks as "limited domains".

Some people have concerns about splintering of the Internet along political or linguistic boundaries by mechanisms that block the free flow of information across the network. That is not the topic of this document, which does not discuss filtering mechanisms and does not apply to protocols that are designed for use across the whole Internet. It is only concerned with domains that have specific technical requirements.

The word "domain" in this document does not refer to naming domains in the DNS, although in some cases a limited domain might incidentally be congruent with a DNS domain.

The requirements of limited domains will be different in different scenarios. Policies, default parameters, and the options supported may vary. Also, the style of network management may vary, between a completely unmanaged network, one with fully autonomic management, one with traditional central management, and mixtures of the above. Finally, the requirements and solutions for security and privacy may vary.

This documents analyses and discusses some of the consequences of this trend, and how it impacts the idea of universal interoperability in the Internet. Firstly we list examples of limited domain scenarios and of technical solutions for limited domains, with the main focus being the Internet layer of the protocol stack. Then we develop a taxonomy of the features to be found in limited domains. With this background, we discuss the resulting challenge to the notion that all Internet standards must be universal in scope and applicability. To the contrary, we assert that some protocols need to be specifically limited in their applicability. This implies that the concepts of a limited domain, and of its membership, need to be formalised and supported by secure mechanisms. While this document does not propose a design for such mechanisms, it does outline some resulting functional requirements.

2. Failure Modes in Today’s Internet

Today, the Internet does not have a well-defined concept of limited domains. One result of this is that certain protocols and features fail on certain paths. Previously, this has been analysed in terms of transparency [RFC2775], [RFC4924] or of intrusive middleboxes [RFC3234], [RFC7663], [I-D.dolson-plus-middlebox-benefits]. Unfortunately the problems persist, both in application protocols, and even in very fundamental mechanisms. For example, the Internet is not transparent to IPv6 extension headers [RFC7872], and Path MTU Discovery has been unreliable for many years [RFC2923], [RFC4821]. IP fragmentation is also unreliable

[I-D.bonica-intarea-frag-fragile], and problems in TCP MSS negotiation have been reported [I-D.andrews-tcp-and-ipv6-use-minmtu].

On the security side, the widespread insertion of firewalls at domain boundaries that are perceived by humans but unknown to protocols results in arbitrary failure modes as far as the application layer is concerned.

This situation is not acceptable, so it seems that a new approach is needed.

3. Examples of Limited Domain Requirements

This section describes various examples where limited domain requirements can easily be identified, either based on an application scenario or on a technical imperative. It is of course not a complete list, and it is presented in an arbitrary order, loosely from smaller to bigger.

1. A home network. It will be unmanaged, constructed by a non-specialist, and will possibly include wiring errors such as physical loops. It must work with devices "out of the box" as shipped by their manufacturers and must create adequate security by default. Remote access may be required. The requirements and applicable principles are summarised in [RFC7368].

2. A small office network. This is sometimes very similar to a home network, if whoever is in charge has little or no specialist knowledge, but may have differing security and privacy requirements. In other cases it may be professionally constructed using recommended products and configurations, but operate unmanaged. Remote access may be required.

3. A vehicle network. This will be designed by the vehicle manufacturer but may include devices added by the vehicle’s owner or operator. Parts of the network will have demanding performance and reliability requirements with implications for human safety. Remote access may be required to certain functions, but absolutely forbidden for others. Communication with other vehicles, roadside infrastructure, and external data sources will be required. See [I-D.ietf-ipwave-vehicular-networking] for a survey of use cases.

4. Supervisory Control And Data Acquisition (SCADA) networks, and other hard real time networks. These will exhibit specific technical requirements, including tough real-time performance targets. See for example [I-D.ietf-detnet-use-cases] for

numerous use cases. An example is a building services network. This will be designed specifically for a particular building, but using standard components. Additional devices may need to be added at any time. Parts of the network may have demanding reliability requirements with implications for human safety. Remote access may be required to certain functions, but absolutely forbidden for others.

5. Sensor networks. The two preceding cases will all include sensors, but some networks may be specifically limited to sensors and the collection and processing of sensor data. They may be in remote or technically challenging locations and installed by non-specialists.

6. Internet of Things (IoT) networks. While this term is very flexible and covers many innovative types of network, including ad hoc networks that are formed spontaneously, it seems reasonable to expect that IoT edge networks will have special requirements and protocols that are useful only within a specific domain, and that these protocols cannot, and for security reasons should not, run over the Internet as a whole.

7. An important subclass of IoT networks consists of constrained networks [RFC7228] in which the nodes are limited in power consumption and communications bandwidth, and are therefore limited to using very frugal protocols.

8. Delay tolerant networks may consist of domains that are relatively isolated and constrained in power (e.g. deep space networks) and are connected only intermittently to the outside, with a very long latency on such connections [RFC4838]. Clearly the protocol requirements and possibilities are very specialised in such networks.

9. "Traditional" enterprise and campus networks, which may be spread over many kilometres and over multiple separate sites, with multiple connections to the Internet. Interestingly, the IETF appears never to have analysed this long-established class of networks in a general way, except in connection with IPv6 deployment (e.g. [RFC7381]).

10. Data centres and hosting centres, or distributed services acting as such centres. These will have high performance, security and privacy requirements and will typically include large numbers of independent "tenant" networks overlaid on shared infrastructure.

11. Content Delivery Networks (CDNs), comprising distributed data centres and the paths between them, spanning thousands of kilometres, with numerous connections to the Internet.

12. Massive Web Service Provider Networks. This is a small class of networks with well known trademarked names, combining aspects of distributed enterprise networks, data centres and CDNs. They have their own international networks bypassing the generic carriers. Like CDNs, they have numerous connections to the Internet, typically offering a tailored service in each economy.

Three other aspects, while not tied to specific network types, also strongly depend on the concept of limited domains:

1. Intent Based Networking. In this concept, a network domain is configured and managed in accordance with an abstract policy known as "Intent", to ensure that the network performs as required [I-D.moulchan-nmrg-network-intent-concepts]. Whatever technologies are used to support this, they will be applied within the domain boundary.

2. Many of the above types of network may be extended throughout the Internet by a variety of virtual private network (VPN) techniques. Therefore we may argue that limited domains may overlap each other in an arbitrary fashion by use of virtualization techniques.

3. Network Slicing. A network slice is a virtual network that consists of a managed set of resources carved off from a larger network [I-D.geng-netslices-architecture]. Whatever technologies are used to support slicing, they will require a clear definition of the boundary of a given slice.

While it is clearly desirable to use common solutions, and therefore common standards, wherever possible, it is increasingly difficult to do so while satisfying the widely varying requirements outlined above. However, there is a tendency when new protocols and protocol extensions are proposed to always ask the question "How will this work across the open Internet?" This document suggests that this is not always the right question. There are protocols and extensions that are not intended to work across the open Internet. On the contrary, their requirements and semantics are specifically limited (in the sense defined above).

A common argument is that if a protocol is intended for limited use, the chances are very high that it will in fact be used (or misused) in other scenarios including the so-called open Internet. This is undoubtedly true and means that limited use is not an excuse for bad

design or poor security. In fact, a limited use requirement potentially adds complexity to both the protocol and its security design, as discussed later.

Nevertheless, because of the diversity of limited environments with specific requirements that is now emerging, specific standards will necessarily emerge. There will be attempts to capture each market sector, but the market will demand standardised limited solutions. However, the "open Internet" must remain as the universal method of interconnection. Reconciling these two aspects is a major challenge.

4. Examples of Limited Domain Solutions

This section lists various examples of specific limited domain solutions that have been proposed or defined. It intentionally does not include Layer 2 technology solutions, which by definition apply to limited domains.

1. Differentiated Services. This mechanism [RFC2474] allows a network to assign locally significant values to the 6-bit Differentiated Services Code Point field in any IP packet. Although there are some recommended codepoint values for specific per-hop queue management behaviours, these are specifically intended to be domain-specific codepoints with traffic being classified, conditioned and re-marked at domain boundaries (unless there is an inter-domain agreement that makes re-marking unnecessary).

2. Network function virtualisation. As described in [I-D.irtf-nfvrg-gaps-network-virtualization], this general concept is an open research topic, in which virtual network functions are orchestrated as part of a distributed system. Inevitably such orchestration applies to an administrative domain of some kind, even though cross-domain orchestration is also a research area.

3. Service Function Chaining (SFC). This technique [RFC7665] assumes that services within a network are constructed as sequences of individual functions within a specific SFC-enabled domain. As that RFC states: "Specific features may need to be enforced at the boundaries of an SFC-enabled domain, for example to avoid leaking SFC information". A Network Service Header (NSH) [RFC8300] is used to encapsulate packets flowing through the service function chain: "The intended scope of the NSH is for use within a single provider’s operational domain."

4. Firewall and Service Tickets (FAST). Such tickets would accompany a packet to claim the right to traverse a network or

request a specific network service [I-D.herbert-fast]". They would only be valid within a particular domain.

5. Data Centre Network Virtualization Overlays. A common requirement in data centres that host many tenants (clients) is to provide each one with a secure private network, all running over the same physical infrastructure. [RFC8151] describes various use cases for this, and specifications are under development. These include use cases in which the tenant network is physically split over several data centres, but which must appear to the user as a single secure domain.

6. Segment Routing. This is a technique which "steers a packet through an ordered list of instructions, called segments" [I-D.ietf-spring-segment-routing]. The semantics of these instructions are explicitly local to a segment routing domain or even to a single node. Technically, these segments or instructions are represented as an MPLS label or an IPv6 address, which clearly adds a semantic interpretation to them within the domain.

7. Autonomic Networking. As explained in [I-D.ietf-anima-reference-model], an autonomic network is also a security domain within which an autonomic control plane [I-D.ietf-anima-autonomic-control-plane] is used by service agents. These service agents manage technical objectives, which may be locally defined, subject to domain-wide policy. Thus the domain boundary is important for both security and protocol purposes.

8. Homenet. As shown in [RFC7368], a home networking domain has specific protocol needs that differ from those in an enterprise network or the Internet as a whole. These include the Home Network Control Protocol (HNCP) [RFC7788] and a naming and discovery solution [I-D.ietf-homenet-simple-naming].

9. Creative uses of IPv6 features. As IPv6 enters more general use, engineers notice that it has much more flexibility than IPv4. Innovative suggestions have been made for:

   * The flow label, e.g. [RFC6294], [I-D.fioccola-v6ops-ipv6-alt-mark].

   * Extension headers, e.g. for segment routing [I-D.ietf-6man-segment-routing-header].

   * Meaningful address bits, e.g. [I-D.jiang-semantic-prefix]. Also, segment routing uses IPv6 addresses as segment

identifiers with specific local meanings [I-D.ietf-spring-segment-routing].

All of these suggestions are only viable within a specified domain. The case of the extension header is particularly interesting, since its existence has been a major "selling point" for IPv6, but it is notorious that new extension headers are virtually impossible to deploy across the whole Internet [RFC7045], [RFC7872]. It is worth noting that extension header filtering is considered as an important security issue [I-D.ietf-opsec-ipv6-eh-filtering]. There is considerable appetite among vendors or operators to have flexibility in defining extension headers for use in limited or specialised domains, e.g. [I-D.voyer-6man-extension-header-insertion] and [BIGIP].

10. Deterministic Networking (DetNet). The Deterministic Networking Architecture [I-D.ietf-detnet-architecture] and encapsulation [I-D.ietf-detnet-dp-sol] aim to support flows with extremely low data loss rates and bounded latency, but only within a part of the network that is "DetNet aware". Thus, as for differentiated services above, the concept of a domain is fundamental.

11. Provisioning Domains (PvDs). An architecture for Multiple Provisioning Domains has been defined [RFC7556] to allow hosts attached to multiple networks to learn explicit details about the services provided by each of those networks.

5. Taxonomy of Limited Domains

This section develops a taxonomy for describing limited domains. Several major aspects are considered in this taxonomy:

- The domain as a whole.
- The individual nodes.
- The domain boundary.
- The domain’s topology.
- The domain’s technology.
- How the domain connects to the Internet.
- The security, trust and privacy model.
- Operations.

The following sub-sections analyse each of these aspects.

5.1. The Domain as a Whole

- Why does the domain exist? (e.g., human choice, administrative policy, orchestration requirements, technical requirements)
- If there are special requirements, are they at Layer 2, Layer 3 or an upper layer?
- Is the domain managed by humans or fully autonomic?
- If managed, what style of management applies? (Manual configuration, automated configuration, orchestration?)
- Is there a policy model? (Intent, configuration policies?)
- Does the domain provide controlled or paid service or open access?

5.2. Individual Nodes

- Is a domain member a complete node, or only one interface of a node?
- Are nodes permanent members of a given domain, or are join and leave operations possible?
- Are nodes physical or virtual devices?
- Are virtual nodes general-purpose, or limited to specific functions, applications or users?
- Are nodes constrained (by battery etc)?
- Are devices installed "out of the box" or pre-configured?

5.3. The Domain Boundary

- How is the domain boundary identified or defined?
- Is the domain boundary fixed or dynamic?
- Are boundary nodes special? Or can any node be at the boundary?

5.4. Topology

- Is the domain a subset of a layer 2 or 3 connectivity domain?
- In IP addressing terms, is the domain Link-local, Site-local, or Global?
- Does the domain overlap other domains? (In other words, a node may or may not be allowed to be a member of multiple domains.)
- Does the domain match physical topology, or does it have a virtual (overlay) topology?
- Is the domain in a single building, vehicle or campus? Or is it distributed?
- If distributed, are the interconnections private or over the Internet?
- In IP addressing terms, is the domain Link-local, Site-local, or Global?

5.5. Technology

- In routing terms, what routing protocol(s) are used, or even different forwarding mechanisms (MPLS or other non-IP mechanism)?
- In an overlay domain, what overlay technique is used (L2VPN, L3VPN,...)?
- Are there specific QoS requirements?
- Link latency - normal or long latency links?
- Mobility - are nodes mobile? Is the whole network mobile?
- Which specific technologies, such as those in Section 4, are applicable?

5.6. Connection to the Internet

- Is the Internet connection permanent or intermittent? (Never connected is out of scope.)
- What traffic is blocked, in and out?
- What traffic is allowed, in and out?

- What traffic is transformed, in and out?
- Is secure and privileged remote access needed?
- Does the domain allow unprivileged remote sessions?

5.7. Security, Trust and Privacy Model

- Must domain members be authorized?
- Are all nodes in the domain at the same trust level?
- Is traffic authenticated?
- Is traffic encrypted?
- What is hidden from the outside?

5.8. Operations

- Safety level - does the domain have a critical (human) safety role?
- Reliability requirement - normal or 99.999% ?
- Environment - hazardous conditions?
- Installation - are specialists needed?
- Service visits - easy, difficult, impossible?
- Software/firmware updates - possible or impossible?

5.9. Making use of this taxonomy

This taxonomy could be used to design or analyse a specific type of limited domain. For the present document, it is intended to form a background to the following two sections, concerning the scope of protocols used in limited domains, and mechanisms reuqired to securely define domain membership and properties.

6. The Scope of Protocols in Limited Domains

One consequence of the deployment of limited domains in the Internet is that some protocols will be designed, extended or configured so that they only work correctly between end systems in such domains. This is to some extent encouraged by some existing standards and by the assignment of code points for local or experimental use. In any

case it cannot be prevented. Also, by endorsing efforts such as Service Function Chaining, Segment Routing and Deterministic Networking, the IETF is in effect encouraging such deployments. Furthermore, it seems inevitable, if the "Internet of Things" becomes reality, that millions of edge networks containing completely novel types of node will be connected to the Internet; each one of these edge networks will be a limited domain.

It is therefore appropriate to discuss whether protocols or protocol extensions should sometimes be standardised to interoperate only within a Limited Domain Boundary. Such protocols would not be required to interoperate across the Internet as a whole. Several possibly overlapping scenarios could then arise:

A. If a limited domain is split into two parts connected over the Internet directly at the IP layer (i.e. with no tunnel encapsulating the packets), a limited-domain protocol could be operated between those two parts regardless of its special nature, as long as it respects standard IP formats and is not arbitrarily blocked by firewalls. A simple example is any protocol using a port number assigned to a specific non-IETF protocol.

Such a protocol could reasonably be described as an "inter-domain" protocol because the Internet is transparent to it, even if it is meaningless except in the two parts of the limited domain. This is of course nothing new in the Internet architecture.

B. If a limited-domain protocol does not respect standard IP formats (for example, if it includes a non-standard IPv6 extension header), it could not be operated between two parts of a domain split at the IP layer.

Such a protocol could reasonably be described as an "intra-domain" protocol, and the Internet is opaque to it.

C. If a limited-domain protocol is clearly specified to be invalid outside its domain of origin, neither scenario A nor B applies. The two domains need to be unified as a single virtual domain. For example, an encapsulating tunnel between the parts of the split domain could be used. Also, nodes at the domain boundary must drop all packets using the limited-domain protocol.

D. If a limited-domain protocol has domain-specific variants, such that implementations in different domains could not interoperate if those domains were unified by some mechanism, the protocol is not interoperable in the normal sense. If two domains using it were merged, the protocol might fail unpredictably. A simple example is any protocol using a port number assigned for

experimental use. Such a protocol usually also falls into scenario C.

To provide an existing example, consider Differentiated Services [RFC2474]. A packet containing any value whatever in the 6 bits of the Differentiated Service Code Point (DSCP) is well-formed and falls into scenario A. However, because the semantics of DSCP values are locally significant, the packet also falls into scenario D. In fact, differentiated services are only interoperable across domain boundaries if there is a corresponding agreement between the operators; otherwise a specific gateway function is required for meaningful interoperability. Much more detailed discussion is to be found in [RFC2474] and [RFC8100].

To provide a provocative example, consider the proposal in [I-D.voyer-6man-extension-header-insertion] that the restrictions in [RFC8200] should be relaxed to allow IPv6 extension headers to be inserted on the fly in IPv6 packets. If this is done in such a way that the affected packets can never leave the specific limited domain in which they were modified, scenario C applies. If the semantic content of the inserted headers is locally defined, scenario D also applies. In neither case is the Internet disturbed.

We conclude that it is reasonable to explicitly define limited-domain protocols, either as standards or as proprietary mechanisms, as long as they describe which of the above scenarios apply and they clarify how the domain is defined. As long as all relevant standards are respected outside the domain boundary, a well-specified limited-domain protocol is not harmful to the Internet. However, as described in the next section, mechanisms are needed to support domain membership operations.

7. Functional Requirements of Limited Domains

As the preceding taxonomy shows, there are very numerous aspects to a domain, so the common features are not immediately obvious. It would be possible, but tedious, to apply the taxonomy to each of the domain types described in Section 3. However, we can deduce some generally required features and functions without doing so.

Firstly, if we drew a topology map, any domain -- virtual or physical -- will have a well defined boundary between "inside" and "outside". However, that boundary in itself has no technical meaning. What matters in reality is whether a node is a member of the domain, and whether it is at the boundary between the domain and the rest of the Internet. Thus the boundary in itself does not need to be identified. However, a sending node needs to know whether it is sending to an inside or outside destination; a receiving node needs

to know whether a packet originated inside or outside; and a boundary node needs to know which of its interfaces are inward-facing or outward-facing. It is irrelevant whether the interfaces involved are physical or virtual.

With this perspective, we can list some general functional requirements. An underlying assumption here is that domain membership operations should be cryptographically secured; a domain without such security cannot be reliably protected from attack.

1. Domain Identity. A domain must have a unique and verifiable identifier; effectively this should be a public key for the domain. Without this, there is no way to secure domain operations and domain membership. The holder of the corresponding private key becomes the trust anchor for the domain.

2. Node Eligibility. It must be possible for a node to determine which domain(s) it can potentially join, and on which interface(s).

3. Secure Enrolment. A node must be able to enrol in a given domain via secure node identfication and to acquire relevant security credentials (authorization) for operations within the domain. If a node has multiple physical or virtual interfaces, they may require to be individually enrolled.

4. Withdrawal. A node must be able to cancel enrolment in a given domain.

5. Dynamic Membership. Optionally, a node should be able temporarily leave or rejoin a domain (i.e. enrolment is persistent but membership is intermittent).

6. Role, implying authorization to perform a certain set of actions. A node must have a verifiable role. In the simplest case, the choices of role are "interior node" and "boundary node". In a boundary node, individual interfaces may have different roles, e.g. "inward facing" and "outward facing".

7. Verify Peer. A node must be able to verify whether another node is a member of the domain.

8. Verify Role. A node must be able to learn the verified role of another node. In particular, it must be possible for a node to find boundary nodes (interfacing to the Internet).

9. Domain Data. In a domain with management requirements, it must be possible for a node to acquire domain policy and/or domain configuration data. This would include, for example, filtering policy to ensure that inappropriate packets do not leave the domain.

These requirements could form the basis for further analysis and solution design.

8. Security Considerations

Clearly, the boundary of a limited domain will almost always also act as a security boundary. In particular, it will serve as a trust boundary, and as a boundary of authority for defining capabilities. Within the boundary, limited-domain protocols or protocol features will be useful, but they will be meaningless if they enter or leave the domain.

The security model for a limited-scope protocol must allow for the boundary, and in particular for a trust model that changes at the boundary. Typically, credentials will need to be signed by a domain-specific authority.

9. IANA Considerations

This document makes no request of the IANA.

10. Contributors

Sheng Jiang made important contributions to this document.

11. Acknowledgements

Useful comments were received from Amelia Andersdotter, Edward Birrane, Ron Bonica, Tim Chown, Darren Dukes, Tom Herbert, John Klensin, Michael Richardson, Rick Taylor, Niels ten Oever, and other members of the ANIMA and INTAREA WGs.

12. Informative References

[BIGIP] Li, R., "HUAWEI - Big IP Initiative.", 2018, <https://www.iaria.org/announcements/HuaweiBigIP.pdf>.

[I-D.andrews-tcp-and-ipv6-use-minmtu] Andrews, M., "TCP Fails To Respect IPV6_USE_MIN_MTU", draft-andrews-tcp-and-ipv6-use-minmtu-04 (work in progress), October 2015.

[I-D.bonica-intarea-frag-fragile]
Bonica, R., Baker, F., Huston, G., Hinden, R., Troan, O., and F. Gont, "IP Fragmentation Considered Fragile", draft-bonica-intarea-frag-fragile-03 (work in progress), July 2018.

[I-D.dolson-plus-middlebox-benefits]
Dolson, D., Snellman, J., Boucadair, M., and C. Jacquenet, "Beneficial Functions of Middleboxes", draft-dolson-plus-middlebox-benefits-03 (work in progress), March 2017.

[I-D.fioccola-v6ops-ipv6-alt-mark]
Fioccola, G., Velde, G., Cociglio, M., and P. Muley, "IPv6 Performance Measurement with Alternate Marking Method", draft-fioccola-v6ops-ipv6-alt-mark-01 (work in progress), June 2018.

[I-D.geng-netslices-architecture]
67, 4., Dong, J., Bryant, S., kiran.makhijani@huawei.com, k., Galis, A., Foy, X., and S. Kuklinski, "Network Slicing Architecture", draft-geng-netslices-architecture-02 (work in progress), July 2017.

[I-D.herbert-fast]
Herbert, T., "Firewall and Service Tickets", draft-herbert-fast-03 (work in progress), September 2018.

[I-D.ietf-6man-segment-routing-header]
Filsfils, C., Previdi, S., Leddy, J., Matsushima, S., and d. daniel.voyer@bell.ca, "IPv6 Segment Routing Header (SRH)", draft-ietf-6man-segment-routing-header-14 (work in progress), June 2018.

[I-D.ietf-anima-autonomic-control-plane]
Eckert, T., Behringer, M., and S. Bjarnason, "An Autonomic Control Plane (ACP)", draft-ietf-anima-autonomic-control-plane-18 (work in progress), August 2018.

[I-D.ietf-anima-reference-model]
Behringer, M., Carpenter, B., Eckert, T., Ciavaglia, L., and J. Nobre, "A Reference Model for Autonomic Networking", draft-ietf-anima-reference-model-08 (work in progress), October 2018.

[I-D.ietf-detnet-architecture]
Finn, N., Thubert, P., Varga, B., and J. Farkas, "Deterministic Networking Architecture", draft-ietf-detnet-architecture-08 (work in progress), September 2018.

[I-D.ietf-detnet-dp-sol]
Korhonen, J., Andersson, L., Jiang, Y., Finn, N., Varga, B., Farkas, J., Bernardos, C., Mizrahi, T., and L. Berger, "DetNet Data Plane Encapsulation", draft-ietf-detnet-dp-sol-04 (work in progress), March 2018.

[I-D.ietf-detnet-use-cases]
Grossman, E., "Deterministic Networking Use Cases", draft-ietf-detnet-use-cases-19 (work in progress), October 2018.

[I-D.ietf-homenet-simple-naming]
Lemon, T., Migault, D., and S. Cheshire, "Simple Homenet Naming and Service Discovery Architecture", draft-ietf-homenet-simple-naming-02 (work in progress), July 2018.

[I-D.ietf-ipwave-vehicular-networking]
Jeong, J., "IP Wireless Access in Vehicular Environments (IPWAVE): Problem Statement and Use Cases", draft-ietf-ipwave-vehicular-networking-04 (work in progress), July 2018.

[I-D.ietf-opsec-ipv6-eh-filtering]
Gont, F. and W. LIU, "Recommendations on the Filtering of IPv6 Packets Containing IPv6 Extension Headers", draft-ietf-opsec-ipv6-eh-filtering-06 (work in progress), July 2018.

[I-D.ietf-spring-segment-routing]
Filsfils, C., Previdi, S., Ginsberg, L., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing Architecture", draft-ietf-spring-segment-routing-15 (work in progress), January 2018.

[I-D.irtf-nfvrg-gaps-network-virtualization]
Bernardos, C., Rahman, A., Zuniga, J., Contreras, L., Aranda, P., and P. Lynch, "Network Virtualization Research Challenges", draft-irtf-nfvrg-gaps-network-virtualization-10 (work in progress), September 2018.

[I-D.jiang-semantic-prefix]
Jiang, S., Qiong, Q., Farrer, I., Bo, Y., and T. Yang, "Analysis of Semantic Embedded IPv6 Address Schemas", draft-jiang-semantic-prefix-06 (work in progress), July 2013.

[I-D.moulchan-nmrg-network-intent-concepts]
Sivakumar, K. and M. Chandramouli, "Concepts of Network Intent", draft-moulchan-nmrg-network-intent-concepts-00 (work in progress), October 2017.

[I-D.voyer-6man-extension-header-insertion]
daniel.voyer@bell.ca, d., Leddy, J., Filsfils, C., Dukes, D., Previdi, S., and S. Matsushima, "Insertion of IPv6 Segment Routing Headers in a Controlled Domain", draft-voyer-6man-extension-header-insertion-04 (work in progress), June 2018.

[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>.

[RFC2775] Carpenter, B., "Internet Transparency", RFC 2775, DOI 10.17487/RFC2775, February 2000, <https://www.rfc-editor.org/info/rfc2775>.

[RFC2923] Lahey, K., "TCP Problems with Path MTU Discovery", RFC 2923, DOI 10.17487/RFC2923, September 2000, <https://www.rfc-editor.org/info/rfc2923>.

[RFC3234] Carpenter, B. and S. Brim, "Middleboxes: Taxonomy and Issues", RFC 3234, DOI 10.17487/RFC3234, February 2002, <https://www.rfc-editor.org/info/rfc3234>.

[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>.

[RFC4838] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., and H. Weiss, "Delay-Tolerant Networking Architecture", RFC 4838, DOI 10.17487/RFC4838, April 2007, <https://www.rfc-editor.org/info/rfc4838>.

[RFC4924] Aboba, B., Ed. and E. Davies, "Reflections on Internet Transparency", RFC 4924, DOI 10.17487/RFC4924, July 2007, <https://www.rfc-editor.org/info/rfc4924>.

[RFC6294] Hu, Q. and B. Carpenter, "Survey of Proposed Use Cases for the IPv6 Flow Label", RFC 6294, DOI 10.17487/RFC6294, June 2011, <https://www.rfc-editor.org/info/rfc6294>.

[RFC7045] Carpenter, B. and S. Jiang, "Transmission and Processing of IPv6 Extension Headers", RFC 7045, DOI 10.17487/RFC7045, December 2013, <https://www.rfc-editor.org/info/rfc7045>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

[RFC7368] Chown, T., Ed., Arkko, J., Brandt, A., Troan, O., and J. Weil, "IPv6 Home Networking Architecture Principles", RFC 7368, DOI 10.17487/RFC7368, October 2014, <https://www.rfc-editor.org/info/rfc7368>.

[RFC7381] Chittimaneni, K., Chown, T., Howard, L., Kuarsingh, V., Pouffary, Y., and E. Vyncke, "Enterprise IPv6 Deployment Guidelines", RFC 7381, DOI 10.17487/RFC7381, October 2014, <https://www.rfc-editor.org/info/rfc7381>.

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

[RFC7663] Trammell, B., Ed. and M. Kuehlewind, Ed., "Report from the IAB Workshop on Stack Evolution in a Middlebox Internet (SEMI)", RFC 7663, DOI 10.17487/RFC7663, October 2015, <https://www.rfc-editor.org/info/rfc7663>.

[RFC7665] Halpern, J., Ed. and C. Pignataro, Ed., "Service Function Chaining (SFC) Architecture", RFC 7665, DOI 10.17487/RFC7665, October 2015, <https://www.rfc-editor.org/info/rfc7665>.

[RFC7788] Stenberg, M., Barth, S., and P. Pfister, "Home Networking Control Protocol", RFC 7788, DOI 10.17487/RFC7788, April 2016, <https://www.rfc-editor.org/info/rfc7788>.

[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>.

[RFC8100] Geib, R., Ed. and D. Black, "Diffserv-Interconnection Classes and Practice", RFC 8100, DOI 10.17487/RFC8100, March 2017, <https://www.rfc-editor.org/info/rfc8100>.

[RFC8151] Yong, L., Dunbar, L., Toy, M., Isaac, A., and V. Manral, "Use Cases for Data Center Network Virtualization Overlay Networks", RFC 8151, DOI 10.17487/RFC8151, May 2017, <https://www.rfc-editor.org/info/rfc8151>.

[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>.

[RFC8300] Quinn, P., Ed., Elzur, U., Ed., and C. Pignataro, Ed., "Network Service Header (NSH)", RFC 8300, DOI 10.17487/RFC8300, January 2018, <https://www.rfc-editor.org/info/rfc8300>.

Appendix A. Change log [RFC Editor: Please remove]

draft-carpenter-limited-domains-00, 2018-06-11:

Initial version

draft-carpenter-limited-domains-01, 2018-07-01:

Minor terminology clarifications

draft-carpenter-limited-domains-02, 2018-08-03:

Additions following IETF102 discussions

Updated authorship/contributors

draft-carpenter-limited-domains-03, 2018-09-12:

First draft of taxonomy

Editorial improvements

draft-carpenter-limited-domains-04, 2018-10-14:

Reorganized section 3

Newly written sections 6 and 7

Editorial improvements

Authors' Addresses

Brian Carpenter
Department of Computer Science
University of Auckland
PB 92019
Auckland 1142
New Zealand

Email: brian.e.carpenter@gmail.com

Bing Liu
Huawei Technologies
Q14, Huawei Campus
No.156 Beiqing Road
Hai-Dian District, Beijing 100095
P.R. China

Email: leo.liubing@huawei.com

Internet-Draft
Intended status: Standard track
Expires April 4, 2019

T. Herbert
Quantonium

October 1, 2018

Control Messages for Generic UDP Encapsulation
draft-herbert-intarea-gue-ctrl-messages-00

Status of this Memo

Copyright Notice

Abstract

This specification defines a set of basic control messages for Generic UDP Encapsulation (GUE). One pair of messages provides a means to query the GUE capabilities of a peer, another pair defines an echo request and response exchange for testing reachability.

Table of Contents

1 Introduction

This specification describes some basic control messages for Generic UDP Encapsulation (GUE). A capabilities query message and response message are defined for a node to query a peer GUE node for supported capabilities. Echo request and echo reply control messages are defined to verify reachability and measure latency to a GUE peer node.

The capabilities query is used to ascertain the capabilities of a peer for receiving GUE messages. For instance, a node may query a GUE peer to determine what GUE variants it supports, or what flags are supported for GUE variant 0. A capabilities query control message and a capabilities response control message are defined. A response message indicates the capabilities for receiving GUE messages. A node may send a capabilities query to a peer GUE node and based on the response it may subsequently use supported flags, optional extensions, GUE variants, or control messages when sending GUE messages to the peer.

Echo request and response messages are used to test for reachability and liveness of a GUE peer node. A node sends an echo request control message and a peer will respond with an echo reply control message. Upon receiving an echo reply, reachability to the GUE peer node is considered verified. The echo request includes arbitrary data that is reflected by the peer in an echo reply. The echo data may contain a timestamp and identifier to perform round trip latency measurement.

1.1 Requirements Language

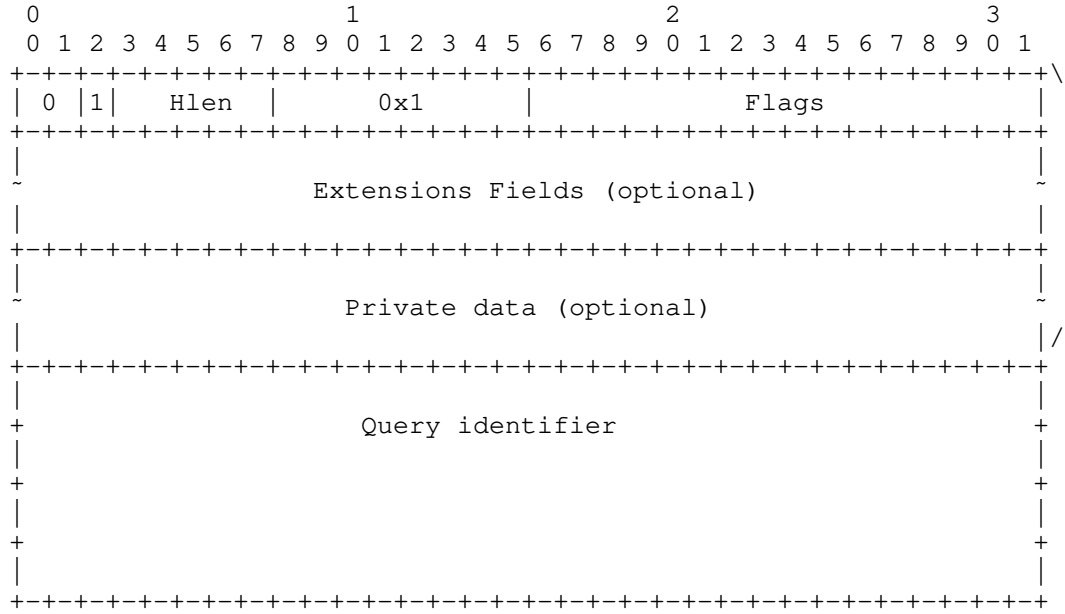
The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2 GUE capabilities query and response messages

This section describes the GUE capabilities query and response messages.

2.1 Capabilities query message

A GUE capabilities query message has the following format:



Pertinent GUE header fields are:

- C bit: Set to 1 to indicate a control message
- Proto/ctype: Set to 0x1 to indicate a capabilities query message

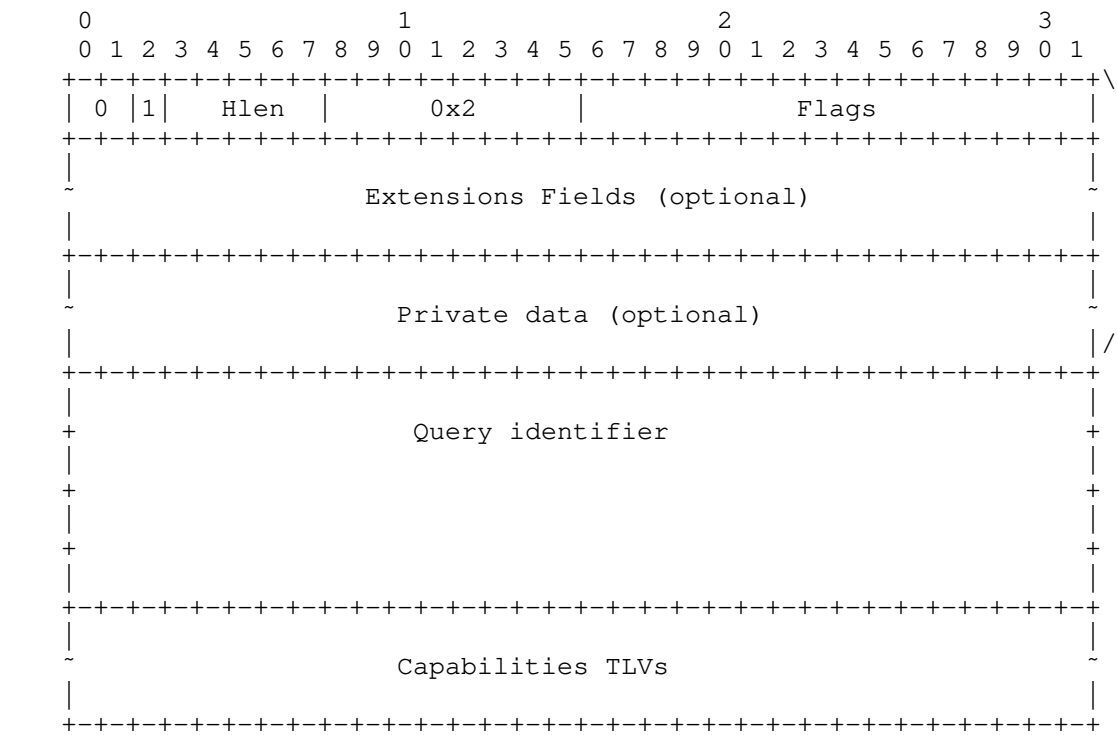
GUE flags, extension fields, and private data SHOULD NOT be used in a capabilities query message.

Control message fields are:

- Query identifier: Used to match queries with responses. This is set to a different non-zero random value in each query.

2.2 Capabilities response message

A GUE capabilities response message has the following format:



Pertinent GUE header fields are:

- C bit: Set to 1 to indicate a control message
- Proto/ctype: Set to 0x2 to indicate a capabilities response message

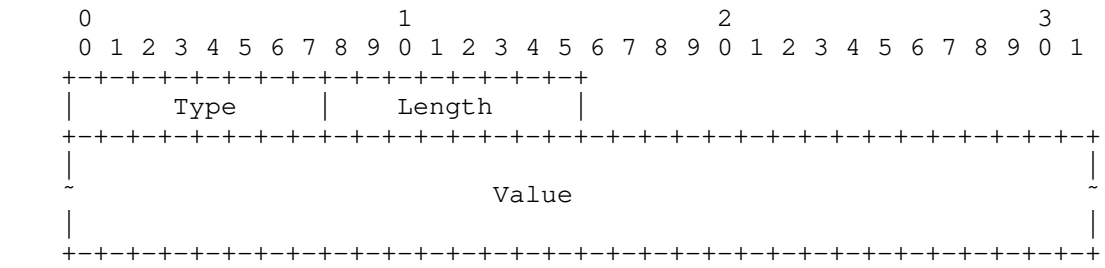
GUE flags, extension fields, and private data SHOULD NOT be used in a capabilities response message.

Control message fields are:

- Query identifier: Reflected value from the capabilities query message
- Capabilities TLVs: A set of Type Length Value (TLV) structures that describe the capabilities of the reporting node

2.3 Capabilities TLV format

Capabilities TLVs have the following format:



Fields:

- Type: Type for TLV. Defined types are described below
- Length: Length in bytes of a TLV Value. Note that this length does not include the two bytes for Type and Length.
- Value: Data for the TLV

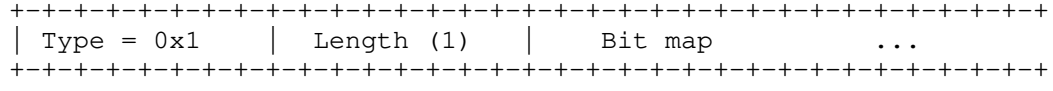
2.4 TLV types

The table below lists the TLVs defined in this document. The "Length" column indicates any required limits on TLVs, and the "Typical Length" column indicates the most useful lengths for the TLV.

Type	Length	Typical Length	Meaning
0			RESERVED
1	variable	1	GUE variants
2	variable	1 to 32	Control message types
3	variable	2 (currently)	GUE flags/extensions
4	variable	1 to 32	Payload transform types
5-126			UNASSIGNED (assignable by IANA)
127-255			User defined

2.4.1 GUE variants

This TLV reports the GUE variants that are support by a node.

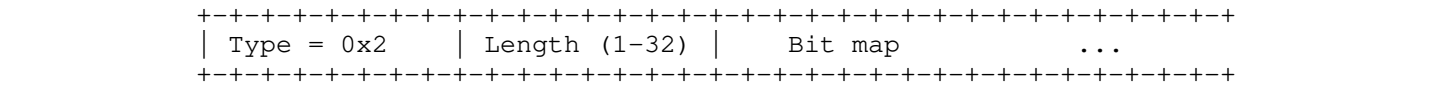


The TLV data is a variable length bit map of supported GUE variants. Bit 0 in the data indicates variant 0 is supported, bit 1 in the data indicates variant 1 is supported, etc. GUE allows up to four variants where variants 0 and 1 have been defined, so only the first four bits in the map are meaningful. If bits are set the map after the fourth bit they are ignored. Similarly, any bytes in the data beyond the first byte are ignored.

Variant 0 must be supported so bit 0 should always be set.

2.4.2 Control message types

This TLV reports the control message types that are supported by a node.

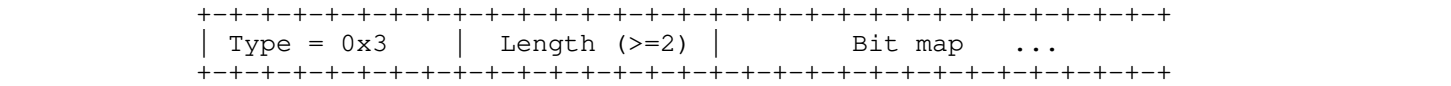


The TLV data is a bit map of supported control message types. Bit 0 in the data indicates control message 0 is supported, bit 1 in the data indicates control message 1 is supported, etc. The range of values for a control message type is 0 to 255, so the maximum useful length of the TLV data is sixteen bytes. If the data length is greater than sixteen bytes then the additional bytes are ignored.

The bit for control message 0 should be set since that value is used to indicate that the payload cannot be parsed as a control message [GUE]. Control message 1 should also be marked as supported given that fact the capabilities represented in the TLV are sent in response to a capabilities query control message which has type of 1.

2.4.3 GUE flags

This TLV reports the GUE flags that are supported by a node. In the case that flags refer to option extensions, the TLV indicates support for the extensions.

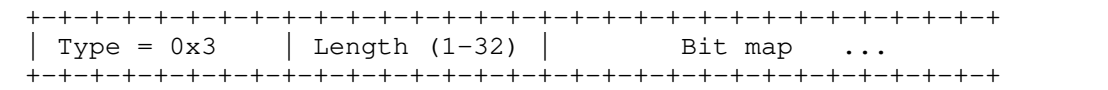


The TLV data is a bit map of supported GUE flags. Bit 0 in the data indicates the flag corresponding to bit 0 of GUE flags is supported, bit 1 indicates the flag for the second bit in GUE flags is supported, etc.

For a multi-bit (paired) flag, the corresponding bits in the bit map are taken to be the maximum value supported for the multi-bit flag. For instance, with a three bit flag, a value of 0x2 indicates flag combinations 0x1 and 0x2 are supported. A value of 0x7 indicates that all seven combinations are supported. If more granularity is needed, for example only values 0x1 and 0x3 are supported, then an additional TLV can be defined to described supported combinations of a multi-bit flag.

2.4.4 Payload transform types

This TLV reports the types of the Payload Transform optional extension that a node supports.



The TLV data is a bit map of supported payload transform types. Bit 0 in the data indicates payload transform type 0 is supported, bit 1 in the data indicates payload transform type 1 is supported, etc. The range of values for a payload transform type is 0 to 255, so the maximum useful length of the TLV data is sixteen bytes. If the data length is greater than sixteen bytes then the additional bytes are ignored.

2.5 Operation

This section describes the operation of capabilities query and response messages.

2.5.1 Sending a capabilities query

A GUE node MAY send a capabilities request to a peer. The request is a well formatted GUE control message. The Query Identifier MUST be set to a non-zero value and SHOULD random. The sender SHOULD save the Query Identifier in a query context to match a response. The sender SHOULD set a timer to receive a response. If no response is received before timeout, then the request context is released.

2.5.2 Receiving a capabilities query

When a node receives a capabilities query it MAY send a response message. The Query Identifier that was received in the query message is reflected in response. The responding node creates the TLVs for capabilities that it wishes to report. A node is not obligated to report all implemented capabilities and may tailor its response per the identity of the requestor. It may withhold reporting of

capabilities for security reasons; for instance, the security option is only useful between two peers if keys are negotiated out of band so indicating support in a capabilities response is not necessary.

2.5.3 Validating a capabilities response message

Upon receiving a capabilities response message, it MUST be validated.

A node SHOULD match the Query Identifier with a recent request that it has sent. If it is unable to match the response to a sent query then the response message SHOULD be dropped. A node MAY choose to match the source address of the response message to the destination address to which it sent the request. Note that GUE does not require addresses to be consistent in the reverse direction. A node may receive a capabilities response sourced from a different address than what it sent the request to; in that case matching the Query Identifier should be sufficient.

If the length of the last TLV exceeds the extent of the packet, then the query response message MUST be dropped. Unknown TLVs are skipped over, as are individual TLVs that have a mismatch in required length or bad data per the requirements of the TLV. TLVs may be sent in any order, may be present more than once in a packet, and the number of TLVs in a message is only limited by packet size. A receiving node may place limits on number or types of TLVs it processes.

2.5.4 Processing a capabilities response

If a valid capabilities response message is received, a node may assume that capabilities indicated in the TLVs are supported by the peer. The node can send GUE packets using those capabilities with the expectation that the peer node will process them. If a capability isn’t indicated as being supported, then a node SHOULD assume its peer doesn’t support the capability and not use it. A node MAY have other information (e.g. out of band configuration) that a peer does support a capability, in which case the capability could be used.

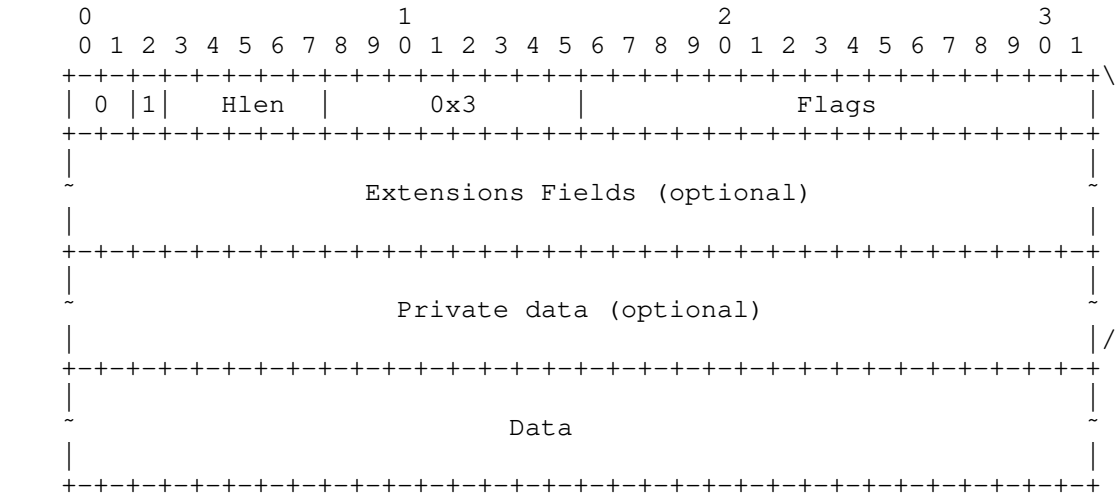
A capabilities query and response exchange is not a protocol negotiation, nor does it establish explicit connection-like state. The reported capabilities should be considered as advisory, and the attained information may be valid for a limited time. It is possible that a node may change its supported capabilities, may refer to a virtual IP address (VIP) where backend nodes support different capabilities, or the address for a peer is reassigned to a node that doesn’t support the same capabilities. A node MAY resend capabilities queries to a destination if it suspects that the supported capabilities might change. The echo request and reply mechanism can also be used to test that reported capabilities are supported.

3 Echo request and reply messages

This section describes the GUE echo request and echo response control messages.

3.1 Echo request

A GUE echo request message has the following format:



Pertinent GUE header fields are:

- C bit: Set to 1 to indicate a control message
- Proto/ctype: Set to 0x3 to indicate an echo request message

GUE flags, extension fields, and private data MAY be used in an echo request.

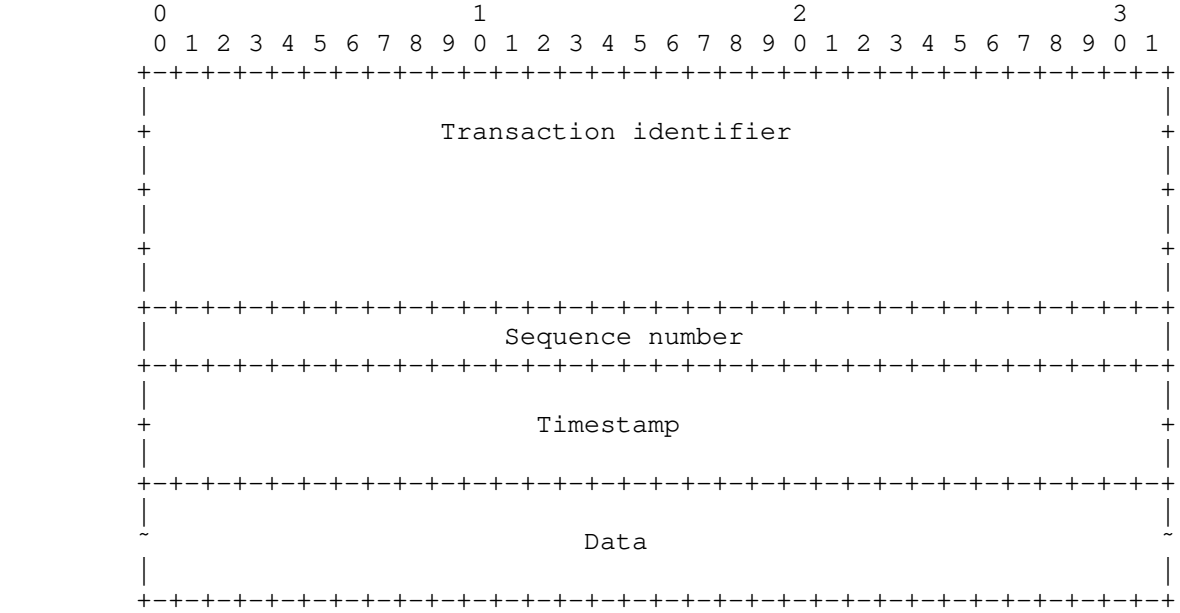
Control message fields are:

- Data: Contains arbitrary data set by the sender. This MAY contain an identifier to match replies with echo requests, and MAY contain a timestamp to measure round trip time.

Note that the data in an echo request is only interpretable by the sender of the echo request. A node receiving an echo request should not attempt to parse the data or interpret it, it should only reflect the data in an echo response.

3.1.1 Optional echo data format

A node MAY use the following format for the echo data. This format includes a transaction identifier, sequence number, and timestamp to facilitate matching replies to requests and measuring round trip latency.



Fields:

- Transaction identifier: Used to match replies with requests. This should be randomly set to a different value for each different destination

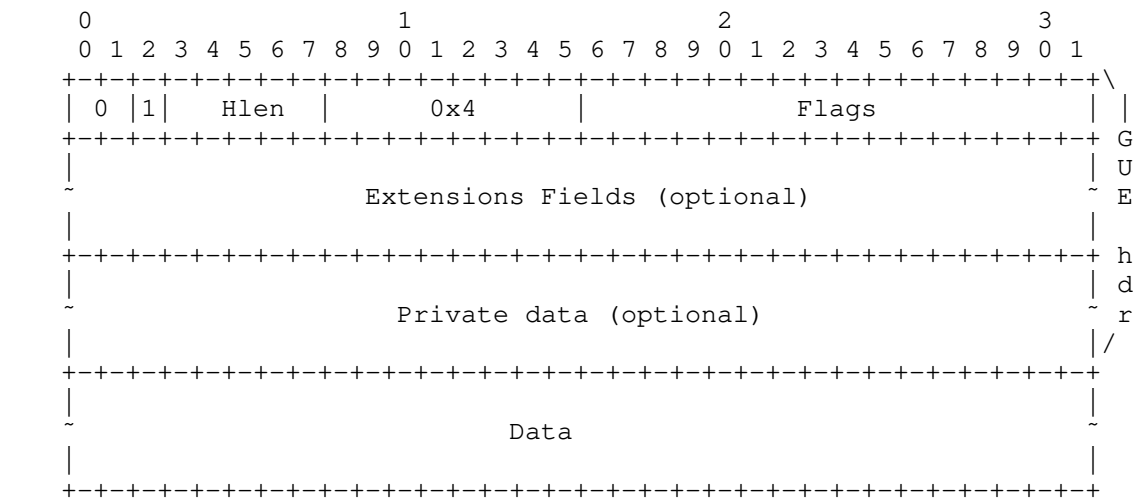
- Sequence number: Monotomically increasing counter for sending multiple echo requests to the same node using the same the transaction identifier

- Timestamp: Timestamp set by the echo request sender and reflected in a echo reply. Normally, this is a value taken from the system clock of the sender. The round trip latency is computed as the time the echo response was received minus the timestamp value received in the echo response. The meaning and units of the timestamp are local to the echo request sender

- Data: Additional data that may be of relevance to the sender

3.2 Echo reply

A GUE echo reply message has the following format:



Pertinent GUE header fields are:

- C bit: Set to 1 to indicate a control message
- Proto/ctype: Set to 0x4 to indicate an echo reply message

GUE flags, extension fields, and private data MAY be used in an echo reply.

Control message fields are:

- Data: Reflected data that was received in an echo request

3.4 Operation

This section describes the operation of echo request and echo reply messages.

3.4.1 Sending an echo request

A node MAY send an echo request message to a peer to determine reachability or measure round trip latency. An echo request is a GUE control message that includes optional data to be reflected by a peer. A node MAY set GUE flags, extensions, and private data-- particularly to test support for these as described below.

If a Transaction Identifier is used in the echo data then the sender SHOULD save it in an echo request context to match to an echo reply. The sender SHOULD set a timer to receive a response. If no response is received before timeout then the echo request context is released.

3.4.2 Receiving an echo request

When a node receives an echo request, an echo response message SHOULD be created and sent back to the source address of the echo request message. A response message SHOULD NOT set any GUE flags, extensions or private data. An exception is if the packet size exceeds MTU then the GUE fragmentation option MAY be used.

3.4.3 Receiving an echo reply

A node SHOULD match a received echo reply to an echo request that it recently sent. If the node sent a Transaction Identifier in an echo request then the value in an echo reply can be matched. Otherwise, an echo reply can be matched to a request based on the source address of the reply message matching the destination address of a recently sent request message. If sequence numbers are present they may be used to track individual echo requests and to report losses.

3.5 Testing GUE capabilities

A node MAY probe the capabilities that a GUE node supports by using the capabilities in an echo request. For instance, a node could set the remote checksum offload option in an echo request. If a corresponding echo reply is received then the node may deduce that its peer supports the feature. This mechanism can be used to verify that capabilities reported in a capabilities response are indeed supported by a peer node.

4 Security considerations

A capabilities response potentially contains detailed information about a system that might be of interest to an attacker. A node MAY choose not to respond to capabilities queries from untrusted nodes, or it may selectively curtail providing information about its capabilities.

Unsolicited capabilities response messages SHOULD NOT be accepted by a node. If a capabilities response is received, then the enclosed Query Identifier SHOULD be matched to a recent query that the node has sent. This is to prevent a attacker from spoofing someone else’s address and reporting random capabilities are supported as an attempted Denial of Service attack.

A node MAY rate limit capabilities response messages and echo reply messages to mitigate Denial of Service attacks.

5 IANA Considerations

5.1 GUE control messages

IANA is requested to assign four values in the registry for the GUE control types:

Control type	Description	Reference
0x1	Capabilities query	This document
0x2	Capabilities response	This document
0x3	Echo request	This document
0x4	Echo reply	This document

5.2 GUE capabilities TLV types

Upon publication, IANA is hereby requested to create a new registry for GUE capabilities TLV types. Initial values of this registry are as listed in Section 2.4.

6 References

6.1 Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[GUE] T. Herbert, L. Yong, and O. Zia, "Generic UDP Encapsulation" draft-ietf-intarea-gue-06

6.2. Informative References

[GUEEXTEN] Herbert, T., Yong, L., and Templin, F., "Extensions for Generic UDP Encapsulation" draft-ietf-intarea-gue-extensions-05

Author's Address

Tom Herbert
Quantonium
4701 Patrick Henry
Santa Clara, CA 95054
US

Email: tom@herbertland.com

Internet-Draft
Intended status: Standard track
Expires March 31, 2019

T. Herbert
Quantonium

September 27, 2018

Generic TCP Encapsulation
draft-herbert-tsvwg-gte-00

Status of this Memo

Copyright Notice

Abstract

This specification describes Generic TCP Encapsulation (GTE) which is a method to encapsulate packets of different IP protocols within TCP data streams. Encapsulating packets in TCP facilitates communications across networks that block or sub-optimally handle non-TCP traffic. GTE is an adaptation of Generic UDP Encapsulation (GUE) to work in the context of TCP. GTE employs the GUE encapsulation format and optional extensions.

Table of Contents

1 Introduction

This specification describes Generic TCP Encapsulation (GTE) which is a general method for encapsulating packets of arbitrary IP protocols within Transport Control Protocol (TCP) [RFC0793] streams. The motivation and basic requirements for encapsulating packets with TCP are described in [TCPENCAP]. The primary motivation is to tunnel packets over networks that either block or treat non-TCP traffic sub-optimally. For instance, a real-time gaming application that uses UDP may chose to encapsulate packets in TCP in order to traverse a stateful firewall that only allows TCP.

GTE is an adaptation of Generic UDP Encapsulation [GUE] to encapsulate packets in a TCP stream. Packets are encapsulated in GTE messages. Each GTE message is composed of a message length, a GUE header variant, and a GUE payload. The message length is needed to demarcate messages in the TCP stream. The TCP data stream is then composed of a sequence of GTE messages. The GUE header and protocol in GTE messages are the same as that defined in GUE. Sender and receiver processing is modified accordingly to take into account the differences between TCP and UDP encapsulation. GTE defines an optimization to compress out the GUE header in certain circumstances.

1.1 Requirements Language

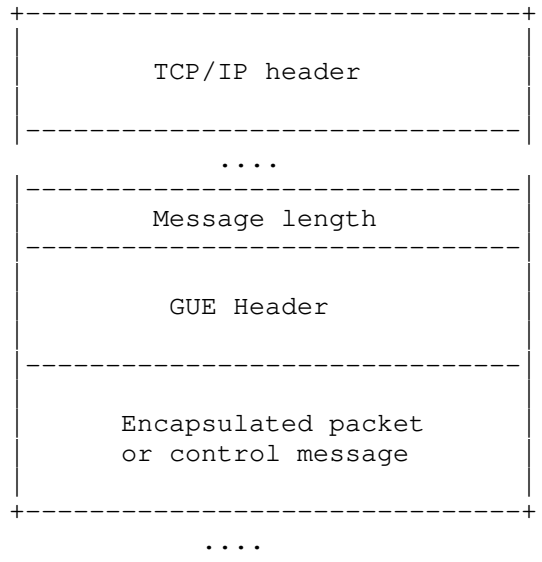
The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2 Encapsulation format

This section describes the encapsulation formats of Generic TCP Encapsulation.

2.1 GTE messages in a TCP stream

A GTE message is encapsulated in a TCP stream. A message is comprised of a message length field, a GUE header of one of the GUE variants, and 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. Encapsulation of GTE message in TCP has the general format:

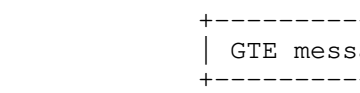


A TCP stream is composed of a sequence of GUE messages. Each message is prefixed by the length of the message for demarcation. Note that there is no on-to-one correspondence between a GTE message and a TCP segment. Multiple GTE messages may be in a single TCP segment, and a single GTE message may span multiple TCP segments. Additionally, there is no assumed alignment of GTE messages within a stream

The GUE header and encapsulation are defined in [GUE]. A GTE message may carry variant 0 or variant 1 GUE packets. The formats are detailed below.

2.2 TCP connections

A GTE stream is composed of GTE message. The message are sequential and per the semantics of TCP they are processed in order.



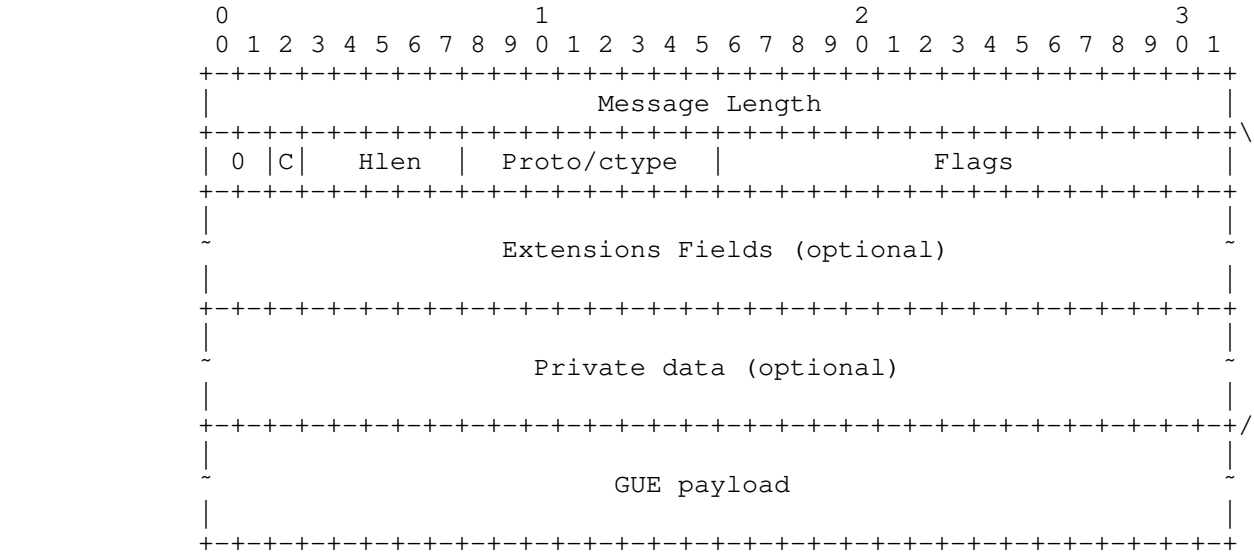
A default TCP port number (suggesting 6080) will be assigned to GTE. A server may listen on this port number for incoming connections. When a connection is established, the data stream is interpreted as a sequence of GTE messages, so the first four bytes in the stream are the length of the first GTE message.

GTE could also use an HTTP connection for the transport. This would be done using the HTTP Upgrade header to specify use of GTE. Specification of this is outside the scope of this document.

Transport Layer Security (TLS) can be used on the TCP stream. In this case, the GTE messages are the plain text before TLS is applied. GTE messages are processed on receive after TLS processing.

2.3 Encapsulation with GUE variant 0

Variant 0 of GUE defines a generic extensible format to encapsulate packets by Internet protocol number. The format of a GTE message with GUE variant 0 is:



Fields:

- Message Length: Length of the GTE message not including the four bytes of this field. The length is the sum of the GUE header length and the length of the encapsulated GUE payload. The size of the message length field may be changed using the Message Length Size control message (section 4.1).

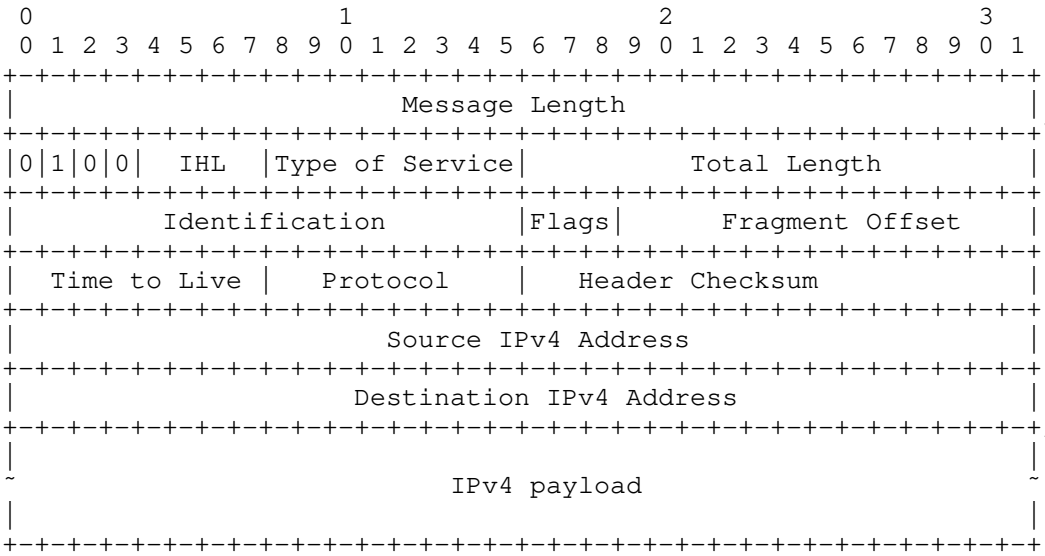
The GUE header fields are specified in [GUE] and GUE extensions are describe in [GUEEXTEN].

2.4 Encapsulation with GUE variant 1

Variant 1 of GUE allows direct encapsulation of IPv4 and IPv6 packets in GUE.

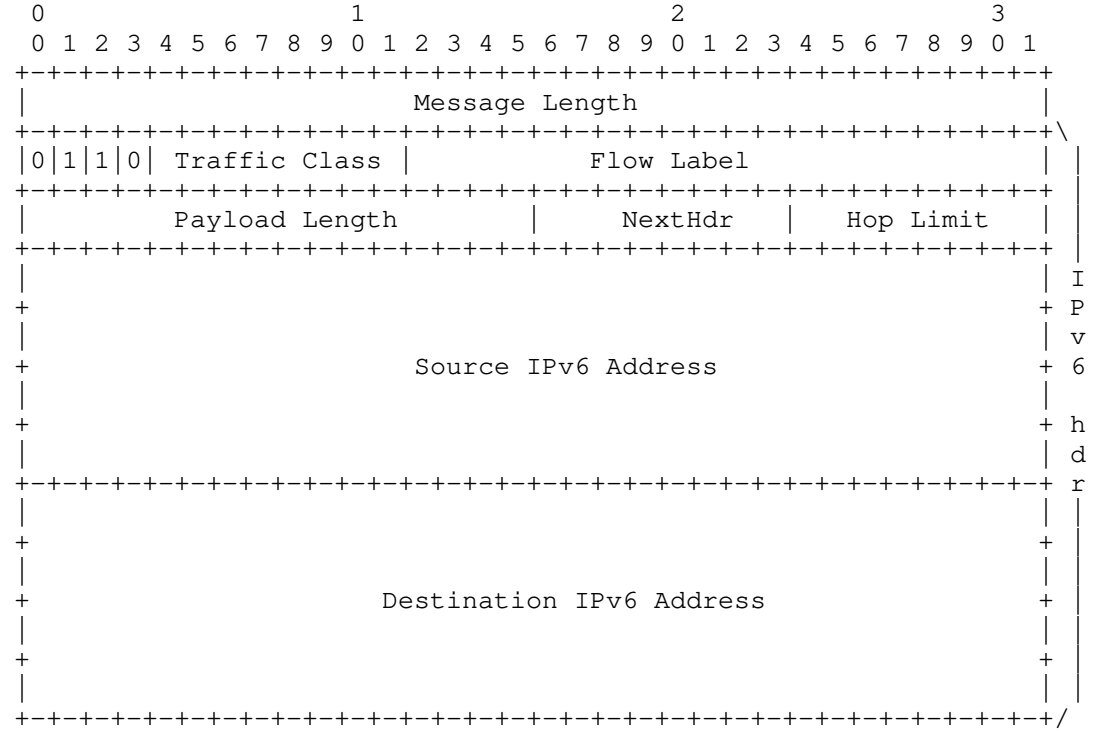
2.4.1 IPv4 direct encapsulation

A GTE message with direct encapsulation of an IPv4 packet has the format:



2.4.2 IPv6 direct encapsulation

A GTE message with direct encapsulation of an IPv6 packet has the format:



3 Operation

This section describes the operation of GTE.

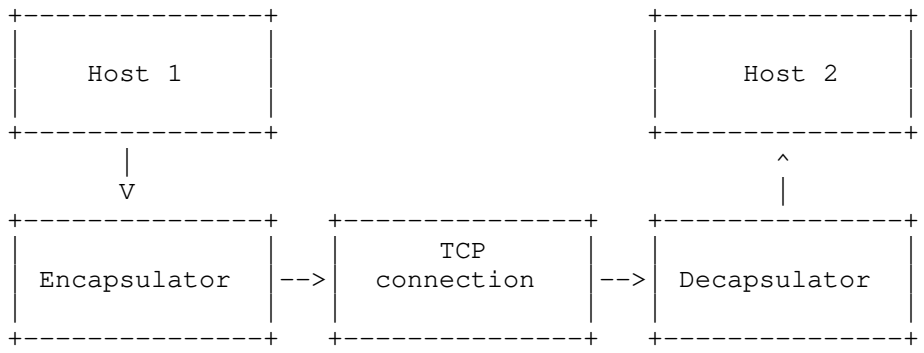
3.1 Encapsulation flavors

GTE can do either network layer of transport layer encapsulation.

3.1.1 Network layer 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 as well as the endpoints of the GTE TCP connection. These could be routers providing tunnels on behalf of communicating hosts.

The figure below illustrates the use of GTE network layer encapsulation between two hosts.

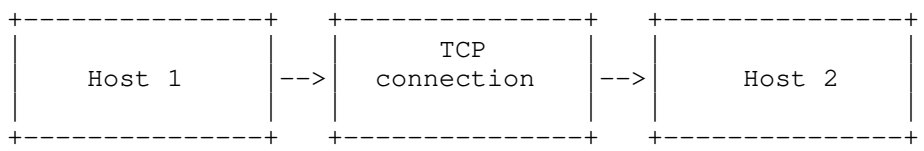


Host 1 is sending packets to Host 2. An encapsulator performs encapsulation of packets from Host 1. These encapsulated packets traverse the network in a TCP stream. At the decapsulator, packets are decapsulated and sent on to Host 2. GTE encapsulation is not required the reverse direction.

3.1.2 Transport layer encapsulation

GTE can encapsulate transport layer packets such as UDP or TCP. The encapsulated packets do not include their own IP header, instead the IP header is inferred from the TCP connection.

The figure below illustrates the use of GTE transport layer encapsulation between two hosts.



When encapsulating transport layer packets, the encapsulator and decapsulator should be co-resident with the hosts. The encapsulated transport layer packets are in a TCP stream. The corresponding IP addresses of the endpoints of the TCP connection are taken to be the endpoints of the encapsulated transport packet. Note that the encapsulated transport layer packet is independent of the encapsulating TCP headers, in particular, port numbers of the outer TCP headers and encapsulated transport headers are independent.

3.2 Encapsulator operation

Encapsulators create GTE data messages, set flags and optional extension fields in the GUE header, and forward packets to a decapsulator by writing messages on a TCP data stream.

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 the peer of the TCP connection.

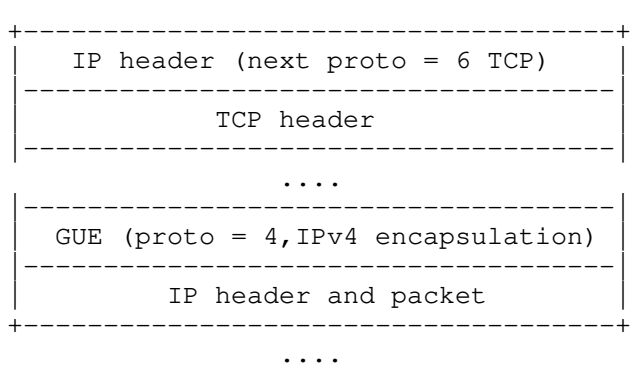
If an encapsulator is tunneling packets -- that is encapsulating packets of layer 2 or layer 3 protocols (e.g. EtherIP, IPIP, or ESP tunnel mode) -- the propagation of network layer information, such as ECN [RFC6040] and diff-serv [RFC2983] should be considered. Since there is no one-to-one mapping between encapsulated packets and the outer packet, the best way to map information may not be obvious. For instance, two IP packets encapsulated in the same TCP segment may have different diff-serv bits. An implementation MAY apply its own configurable heuristics for tunneling of one protocol over another.

3.3 Decapsulator operation

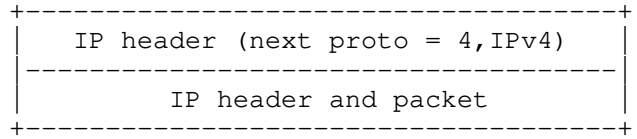
If a complete GTE data message is received on a TCP stream, the payload of the GTE message is logically extracted. An IP packet is created with an IP header that is fabricated from the TCP connection parameters. The next protocol in the IP header is set to the protocol from the proto field in the GUE header. The resulting packet is then resubmitted into the protocol stack to process as though it was received with the protocol in the GUE header.

3.3.1 Receiving network layer encapsulation

Consider that a data message is received where GTE encapsulated an IP packet. In this case, the proto field in the GUE header is set to 4.



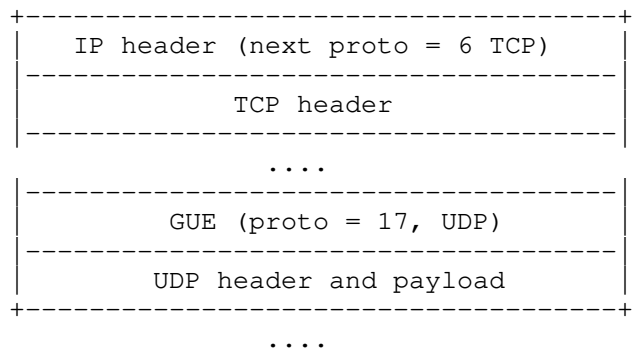
The receiver extracts the IP header and payload, and encapsulates that in an IP packet using IPIP encapsulation. The IP header is derived from the TCP connection or IP header that was present in a received TCP segment. The next protocol field is set to 4. The resultant packet would have the format:



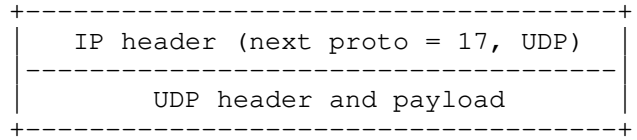
This packet is then resubmitted into the protocol stack to be processed as an IPIP packet.

3.3.2 Receiving transport layer encapsulation

Consider that a data message is received where GTE encapsulates a UDP packet. In this case, the proto field in the GUE header is set to 17 for UDP:



The receiver extracts the UDP header and payload, and formats those in a corresponding UDP/IP packet. The IP header is derived from the TCP connection or IP header that was present in a received TCP segment. The next protocol field is set to 17. The resultant packet would have the format:



This packet is then resubmitted into the protocol stack to be processed as a UDP packet.

3.3.3 Error handling

If a receiver encounters an error in processing the GUE header it SHOULD close the GTE connection and MAY log an error. The possible errors in processing a GUE header are described in [GUE].

3.4 Processing a received control message

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

3.5 NAT interaction

Transport layer encapsulation (section 3.1.2) allows TCP and UDP packets (without IP headers) to be directly encapsulated in GTE. The IP headers for these are derived from those used in the TCP connection. In particular, the pseudo header for an encapsulated transport checksum might cover the IP addresses used in outer TCP packets. When a packet traverses a NAT and the addresses of the TCP packet changes, the checksum for an encapsulated TCP or UDP packet becomes incorrect.

To resolve this problem, the NAT Address Checksum option can be used [GUEEXEN]. A sender computes the one's complement checksum over the IP addresses for the outer TCP connection and sets the value in the Checksum field of the GUE NAT Address Checksum option. When a receiver processes the GTE message, it can compute the required checksum adjustment by taking the difference between checksum over the IP addresses for the connection that it sees and the value it received in the option. If the adjustment is non-zero then that can be added to the computed packet checksum in verification. The exact algorithm is described in [GUEEXTEN].

3.6 Checksum offload of encapsulated transport checksum

Checksum offload is a common technique in Network Interface Cards (NICS) to improve performance [UDPENCAP]. Checksum offload is done on a per packet basis and typically can offload only one checksum in a packet. Several techniques have been implemented to make general use of the capability to effectively offload checksum computation for any number of encapsulated transport checksums in a packet. These techniques include "checksum-unnecessary conversion", Local Checksum Offload (LCO), and Remote Checksum Offload (RCO).

In the case of GTE, packets are encapsulated in a stream so the aforementioned techniques for checksum offload don't directly apply and must be adapted.

3.6.1 Transmit checksum offload

To offload an encapsulated transport layer checksum for transmission, the Remote Checksum Offload (RCO) Option is used [GUEXTEN]. The Checksum Start and Checksum Offset fields in the option are set to point to the starting byte for checksum calculation and the offset where the checksum is to be written. When a receiver processes this field, it will write the derived checksum value at the indicated offset (i.e. the checksum field of an encapsulated transport protocol).

3.6.2 Receive checksum offload

To offload a transport checksum calculation in a received GTE packet, the fact that TCP packets must always include a checksum can be leveraged. Given the TCP payload (or payloads) containing a GTE message, the checksum over the GTE message can be derived by subtracting out the checksum for portions of the payload that aren't part of the GTE message. Once the checksum over the GTE message is determined, that can be used to verify any encapsulated transport layer checksums in the message. Note that this is most efficient when a TCP segment contains only one (or part of one) GTE message.

3.7 GUE extensions

There is no prohibition to using any of the GUE extensions [GUEEXTENS] in GTE. Their usefulness in the context of a TCP stream may vary. The use of NAT Address Checksum and Remote Checksum Offload options are described above. The Fragmentation option is not very useful since TCP provides segmentation and reassembly. Similarly, the GUE checksum option is not useful since the whole message is already covered by the TCP checksum. The Security, Payload Transform, and Alternate Checksum options may be useful to provide security or strong data integrity checks. The Group Identifier might be useful in some cases.

3.8 Surplus area

A an encapsulated packet in GTE may have its own length field such that the encapsulated packet does not take up all the space of the GTE message as indicate by GTE message length. Any bytes beyond the encapsulated packet to the end of message are called "surplus area". Surplus area, for those protocols that have a separate length field, is considered to be reserved. [UDPOPTIONS] is a specification for placing UDP options in surplus area.

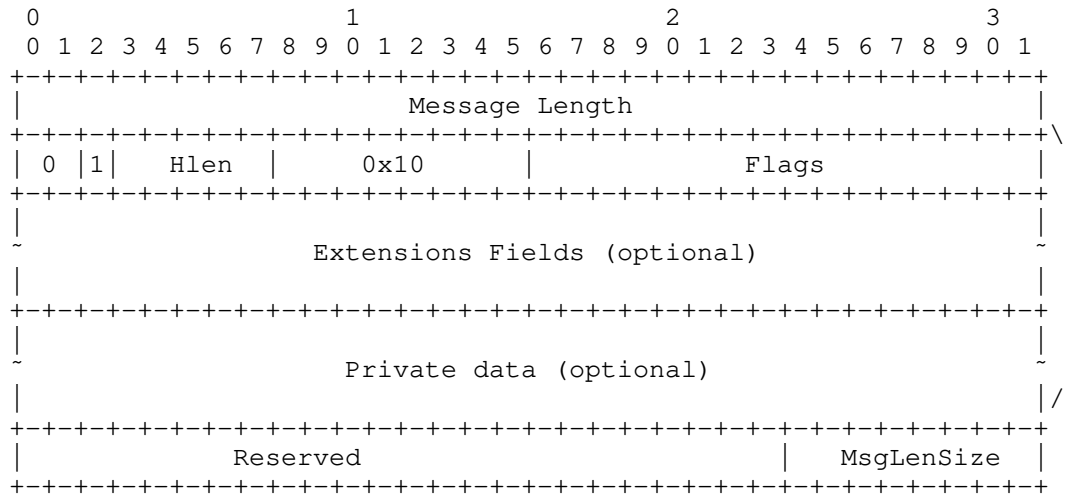
4 GTE control messages

Two GUE control messages are defined to set parameters on a GTE connection. One control message sets the size of the message length field, and the other sets a GUE header template for compressing out the GUE header in GTE messages.

4.1 Message length size

The default message length field in GTE is four bytes. The GUE "Message length size" control message allows the size to be changed for a GTE connection.

The format of the message length size control message is:



Pertinent fields are:

- GUE C bit: Set to 1 to indicate control message
- GUE Proto/ctype field: Set to 0x10 to indicate a message length size control message
- Reserved: Set to zero on transmit, ignored on receive
- MsgLenSize: Number of byte for message length size. Valid values are 1,2,3, or 4

Note that the Reserved and MsgLenSize fields are in the payload of the GUE message. These fields are not included in the GUE header length.

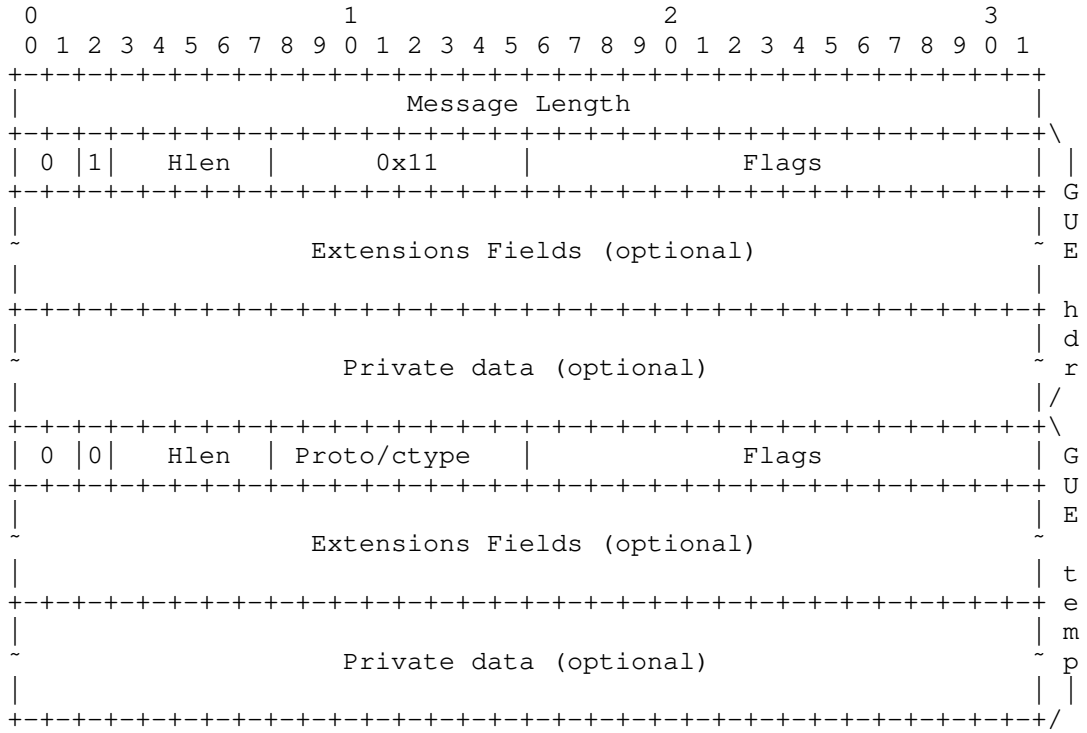
If a received message is too short to include the MsgLenSize field or the MsgLenSize is zero or a value greater then 4, then the connection MUST be closed and an error MAY be logged.

After a message length size control message is received, the size of the message length field in subsequent messages is taken to be the provided value. The message length size control message MAY be sent multiple times to use different sizes for lifetime of the connection.

4.2 GUE header template

An application may use GTE to always tunnel one specific protocol over GTE where the GUE header is identical in all GTE messages. The "GUE header template" control message is defined to compress out the GUE header in such a case. The payload of this control message indicates the GUE header to be applied to all subsequent messages.

The format of the GUE header template control message is:



Pertinent fields in the (outer) GUE header are:

- Variant: Set to 0
- C bit: Set to 1 to indicate control message
- Proto/ctype field: Set to 0x11 to indicate a header template control message

Pertinent fields in the GUE header template GUE are:

- Variant: Must be set to 0
- C bit: Should be zero to indicate a data message
- Proto/ctype: Set to the common IP protocol number for all the messages in the GTE stream

The GUE header template may have flags set, extension fields present, as well as private data.

4.2.1 Header template validation

The format of received GUE template header should be validated like a normal GUE header as described in [GUE]. If the header template is determined to be malformed, then the connection SHOULD be dropped and an error MAY be logged.

The GUE header template must be variant 0. If a GUE header template is received with a variant that is another value the connection MUST be closed and an error MAY be logged.

4.2.2 Optional extensions in header templates

Any GUE extension that may be set in a header template will need to be applicable to all messages.

The Checksum and Alternate Checksum options are not useful in a GUE header template since their input includes payload data. If a proposed header template contains such options the connection SHOULD be dropped and an error MAY be logged.

The Fragmentation option is nonsensical to be in a GUE header template. If a proposed header template contains the Fragmentation option then the connection SHOULD be dropped and an error MAY be logged.

The Security option may be useful in a GUE header template if it does

not include any payload data as input. If the Security option in a header template requires payload data as input, then the connection SHOULD be dropped and an error MAY be logged. If the Security option in a header template doesn't require payload data as input, then it SHOULD be verified (in this case all the input should be contained within the GUE header template). If the Security option fails verification, then the connection SHOULD be dropped and an error MAY be logged.

The Group Identifier, Remote Checksum Offload, NAT Address Checksum and Payload Transform options are valid optional extensions to use in a GUE header template.

4.2.3 Processing received messages

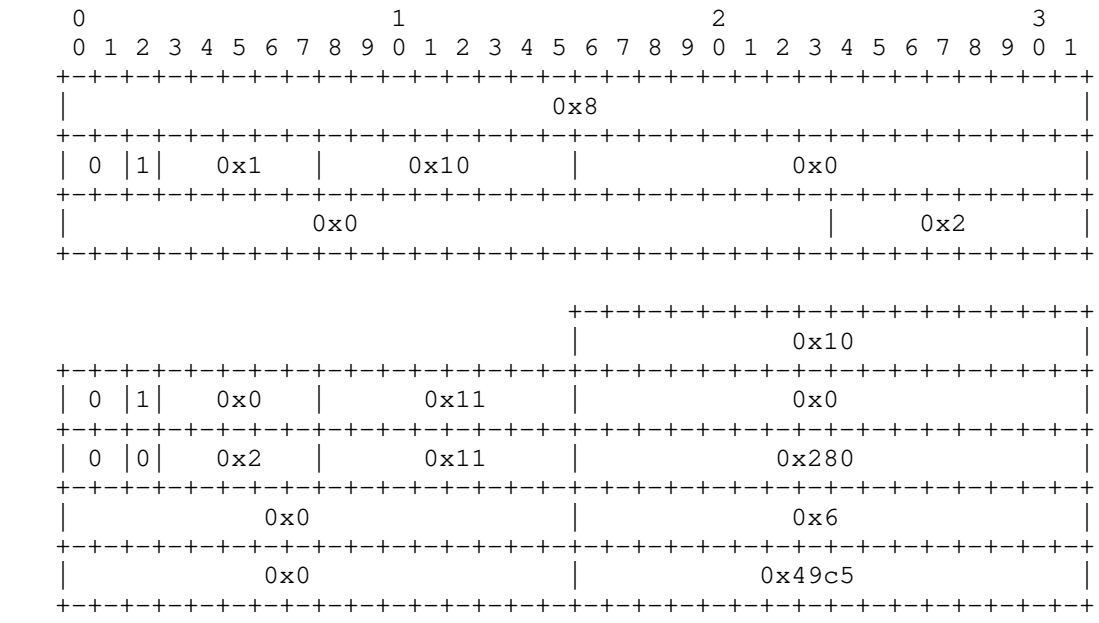
After a GUE header template control message is received, the header template is applied to all subsequent GTE messages. GTE messages will contain a message length followed by the GUE payload. For each received message, the GUE header from the saved template for the connection is logically inserted into the message, and the message is processed as though the GUE header was received.

The GUE header template control message can only be sent once on a connection. All messages following the control message are considered to have the same GUE header and there is no way to send any more GUE control messages on the stream. If both the message length size and the GUE header template are to be set for a GTE connection, then the message length size control message MUST be sent first.

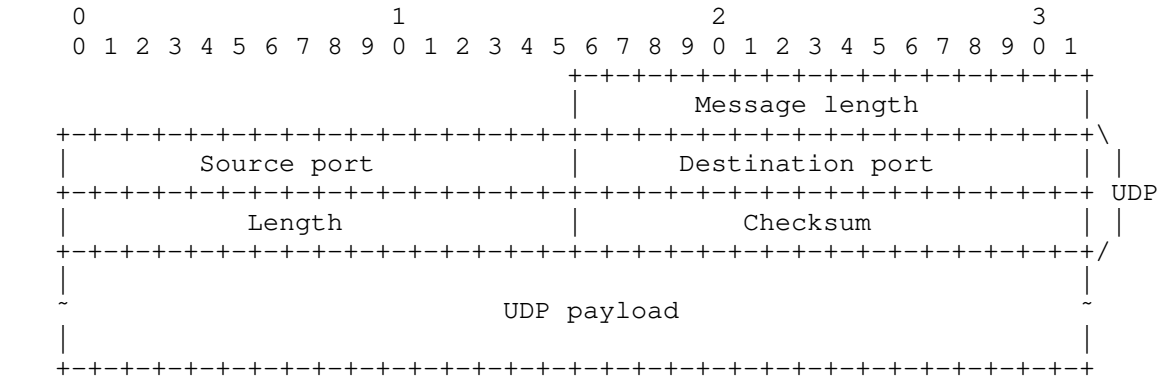
4.3 Example of UDP over GTE

This section provides an example use of the GTE control message to encapsulate a stream of UDP packets over a GTE connection. In this example, the sender uses both the message length size and GUE header template control messages. The message length size is set to two bytes to accommodate the largest UDP packet (65535 bytes). The GUE header template sets the protocol for all messages to be UDP, enables Remote Checksum Offload, and enables the NAT Address Checksum option. In this example, the checksum start and checksum offset field in the Remote Checksum Offload option take values of 0 and 6. The addresses of the connection in this example are 2001:DB8::c099:1:2:0 and 2001:DB8::a92a:7:8:1-- the one's complement sum over those addresses is 0x49c5.

The message length size and the GUE header template control messages would be as follows:



All the messages following these control messages are encapsulated UDP packets in the following format:



Note that for each message, a full UDP/IP packet can be created by deriving an IP header from the TCP connection (the address endpoints of the TCP connection are the same the addresses of the UDP packet). The NAT Address Checksum option can be used to insure that the UDP checksum remains correct if the TCP connection goes through a NAT.

5 Security Considerations

GUE security mechanisms are applicable in GTE. Security considerations for TCP encapsulation are discussed in [TCPENCAP].

6 IANA Considerations

6.1 TCP port number

IANA is requested to assign one TCP port number for Generic TCP Encapsulation. The suggested number is 6080 which is the same as the UDP port number assigned for Generic UDP Encapsulation.

6.2 GUE control types

IANA is requested to assign two values in the registry for the GUE control types:

Control type	Description	Reference
0x10	GTE message length size	This document
0x11	GUE header template	This document

6 Acknowledgements

7 References

7.1 Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[GUE] T. Herbert, L. Yong, and O. Zia, "Generic UDP Encapsulation", draft-ietf-intarea-gue-06

7.2. Informative References

[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>.

[TCPENCAP] T. Pauly and E. Kinnear, "TCP Encapsulation Considerations", draft-pauly-tcp-encapsulation-00

[GUEEXTEN] Herbert, T., Yong, L., and Templin, F., "Extensions for Generic UDP Encapsulation" draft-ietf-intarea-gue-extensions-05

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

[LCO] Cree, E., https://www.kernel.org/doc/Documentation/networking/checksum-offloads.txt

Author's Address

Tom Herbert
Quantonium
4701 Patrick Henry
Santa Clara, CA 95054
US

Email: tom@herbertland.com

Internet Area WG
Internet-Draft
Intended status: Best Current Practice
Expires: April 21, 2019

R. Bonica
Juniper Networks
F. Baker
Unaffiliated
G. Huston
APNIC
R. Hinden
Check Point Software
O. Troan
Cisco
F. Gont
SI6 Networks
October 18, 2018

IP Fragmentation Considered Fragile
draft-ietf-intarea-frag-fragile-02

Abstract

This document describes IP fragmentation and explains how it reduces the reliability of Internet communication.

This document also proposes alternatives to IP fragmentation and provides recommendations for 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 April 21, 2019.

Table of Contents

1. Introduction

Operational experience [Kent] [Huston] [RFC7872] reveals that IP fragmentation reduces the reliability of Internet communication. This document describes IP fragmentation and explains how it reduces the reliability of Internet communication. This document also proposes alternatives to IP fragmentation and provides recommendations for developers and network operators.

While this document identifies issues associated with IP fragmentation, it does not recommend deprecation. Some applications (see Section 6) require IP fragmentation. Furthermore, fragmentation is expected to work in limited domains where security and interoperability issues can be addressed.

Rather than deprecating IP Fragmentation, this document recommends that upper-layer protocols address the problem of fragmentation at their layer, reducing their reliance on IP fragmentation to the greatest degree possible.

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 routers. If a path contains more than one link, the links are connected in series and a router connects each link to the next.

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

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 support a specified MTU (see footnote). IPv6 [RFC8200] requires every link to support an MTU of 1280 bytes or greater. These are called the IPv4 and IPv6 minimum link MTU's.

Likewise, 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:

- The estimate for each IPv4 path is equal to the IPv4 minimum link MTU.
- The estimate for each IPv6 path is equal to the IPv6 minimum link MTU.

While these conservative estimates are guaranteed to be less than or equal to the actual PMTU, 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 estimate of the PMTU between itself and a destination node. In PMTUD, 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 sends 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 its initial value and repeats the procedure described above.

PMTUD has the following characteristics:

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

FOOTNOTE: In IPv4, every 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 791 explicitly states that the IPv4 minimum link MTU is 68 bytes. But for practical purposes, many network operators consider the IPv4 minimum link MTU to be 576 bytes. So, for the purposes of this document, we assume that the IPv4 minimum link MTU is 576 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 indicates 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. Fragmentation Procedures

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

[RFC0791] describes IPv4 fragmentation procedures. An IPv4 packet whose DF-bit is set to one cannot be fragmented. An IPv4 packet whose DF-bit is set to zero can be fragmented by the source node or by any downstream router. When an IPv4 packet is fragmented, all IP options appear in the first fragment, but only options whose "copy" bit is set to one appear in subsequent fragments.

[RFC8200] describes IPv6 fragmentation procedures. An IPv6 packets can be fragmented at the source node only. When an IPv6 packet is fragmented, all extension headers appear in the first fragment, but only per-fragment headers appear in subsequent fragments. Per-fragment headers include the following:

- The IPv6 header.
- The Hop-by-hop Options header (if present)

- The Destination Options header (if present and if it precedes a Routing header)
- The Routing Header (if present)
- The Fragment Header

In both IPv4 and IPv6, the upper-layer header appears in the first fragment only. It does not appear in subsequent fragments.

2.3. Upper-Layer Reliance on IP Fragmentation

Upper-layer protocols can operate in the following modes:

- Do not rely on IP fragmentation.
- Rely on IP fragmentation by the source node only.
- Rely on IP fragmentation by any node.

Upper-layer protocols running over IPv4 can operate in all of the above-mentioned modes. Upper-layer protocols running over IPv6 can operate in the first and second modes only.

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:

- Estimate the PMTU to be equal to the IPv4 or IPv6 minimum link MTU.
- Access the estimate that PMTUD produced.
- Execute PMTUD procedures themselves.
- Execute Packetization Layer PMTUD (PLPMTUD) [RFC4821] [I-D.ietf-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 upper-layer peer and receiving acknowledgements. This strategy differs from PMTUD in that it relies of 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.

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. Reduced Reliability

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

4.1. Policy-Based Routing

IP Fragmentation causes problems for routers that implement policy-based routing.

When a router receives a packet, it identifies the next-hop on route to the packet's destination and forwards the packet to that next-hop. In order to identify the next-hop, the router interrogates a local data structure called the Forwarding Information Base (FIB).

Normally, the FIB contains destination-based entries that map a destination prefix to a next-hop. Policy-based routing allows destination-based and policy-based entries to coexist in the same FIB. A policy-based FIB entry maps multiple fields, drawn from either the IP or transport-layer header, to a next-hop.

Entry	Type	Dest. Prefix	Next Hdr / Dest. Port	Next-Hop
1	Destination-based	2001:db8::1/128	Any / Any	2001:db8::2
2	Policy-based	2001:db8::1/128	TCP / 80	2001:db8::3

Table 1: Policy-Based Routing FIB

Assume that a router maintains the FIB in Table 1. The first FIB entry is destination-based. It maps the a destination prefix (2001:db8::1/128) to a next-hop (2001:db8::2). The second FIB entry is a policy-based. It maps the same destination prefix (2001:db8::1/128) and a destination port (TCP / 80) to a different

next-hop (2001:db8::3). The second entry is more specific than the first.

When the router receives the first fragment of a packet that is destined for TCP port 80 on 2001:db8::1, it interrogates the FIB. Both FIB entries satisfy the query. The router selects the second FIB entry because it is more specific and forwards the packet to 2001:db8::3.

When the router receives the second fragment of the packet, it interrogates the FIB again. This time, only the first FIB entry satisfies the query, because the second fragment contains no indication that the packet is destined for TCP port 80. Therefore, the router selects the first FIB entry and forwards the packet to 2001:db8::2.

Policy-based routing is also known as filter-based-forwarding.

4.2. Network Address Translation (NAT)

IP fragmentation causes problems for Network Address Translation (NAT) devices. When a NAT device detects a new, outbound flow, it maps that flow's source port and IP address to another source port and IP address. Having created that mapping, the NAT device translates:

- The Source IP Address and Source Port on each outbound packet.
- The Destination IP Address and Destination Port on each inbound packet.

A+P [RFC6346] and Carrier Grade NAT (CGN) [RFC6888] are two common NAT strategies. In both approaches the NAT device must virtually reassemble fragmented packets in order to translate and forward each fragment.

Virtual reassembly in the network is problematic, because it is computationally expensive and because it is prone to attacks (Section 4.5).

4.3. Stateless Firewalls

IP fragmentation causes problems for stateless firewalls whose rules include TCP and UDP ports. Because port information is not available in the trailing fragments the firewall is limited to the following options:

- Accept all trailing fragments, possibly admitting certain classes of attack.

- Block all trailing fragments, possibly blocking legitimate traffic.

Neither option is attractive.

This problem does not occur in stateful firewalls.

4.4. Stateless Load Balancers

IP fragmentation causes problems for stateless load balancers. 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:

- IP Source Address.

- IP Destination Address.

- IPv4 Protocol or IPv6 Next Header.

- transport-layer source port.

- transport-layer destination port.

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:

- IP Source Address.

- IP Destination Address.

- 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.

4.5. Security Vulnerabilities

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

- Overlapping fragment attacks [RFC1858][RFC3128][RFC5722]
- Resource exhaustion attacks (such as the Rose Attack)
- Attacks based on predictable fragment identification values [RFC7739]
- 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 procedures described in RFC 1858, RFC 3128 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. Therefore, it can be exploited by resource exhaustion attacks. An attacker can construct a series of fragmented packets, with one fragment missing from each packet so that the reassembly is impossible. 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.

Each IP fragment contains an "Identification" field that destination nodes use to reassemble fragmented packets. Many implementations set the Identification field to a predictable value, thus making it easy for an attacker to forge malicious IP fragments that would cause the reassembly procedure for legitimate packets to fail.

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 (e.g. By

computing all possible outcomes of the fragment reassembly process, at the expense of increased processing requirements).

4.6. Blackholing Due to ICMP Loss

As mentioned in Section 2.3, upper-layer protocols can be configured to rely on PMTUD. Because PMTUD relies upon the network to deliver ICMP PTB messages, those protocols also rely on the networks to deliver ICMP PTB messages.

According to [RFC4890], ICMP PTB messages must not be filtered. However, ICMP PTB delivery is not reliable. It is subject to both transient and persistent loss.

Transient loss of ICMP PTB messages can cause transient black holes. When the conditions contributing to transient loss abate, the network regains its ability to deliver ICMP PTB messages and connectivity between the source and destination nodes is restored. Section 4.6.1 of this document describes conditions that lead to transient loss of ICMP PTB messages.

Persistent loss of ICMP PTB messages can cause persistent black holes. Section 4.6.2 and Section 4.6.3 of this document describe conditions that lead to persistent loss of ICMP PTB messages.

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 from any other source.

4.6.1. Transient Loss

The following factors can contribute to transient loss of ICMP PTB messages:

- Network congestion.
- Packet corruption.
- Transient routing loops.
- ICMP rate limiting.

The effect of rate limiting may be severe, as RFC 4443 recommends strict rate limiting of IPv6 traffic.

4.6.2. Incorrect Implementation of Security Policy

Incorrect implementation of security policy can cause persistent loss of ICMP PTB messages.

Assume that a Customer Premise Equipment (CPE) router implements the following zone-based security policy:

- Allow any traffic to flow from the inside zone to the outside zone.
- Do not allow any traffic to flow from the outside zone to the inside zone unless it is part of an existing flow (i.e., it was elicited by an outbound packet).

When a correct implementation of the above-mentioned security policy receives an ICMP PTB message, it examines the ICMP PTB payload in order to determine whether the original packet (i.e., the packet that elicited the ICMP PTB message) belonged to an existing flow. If the original packet belonged to an existing flow, the implementation allows the ICMP PTB to flow from the outside zone to the inside zone. If not, the implementation discards the ICMP PTB message.

When a incorrect implementation of the above-mentioned security policy receives an ICMP PTB message, it discards the packet because its source address is not associated with an existing flow.

The security policy described above is implemented incorrectly on many consumer CPE routers.

4.6.3. Persistent Loss Caused By Anycast

Anycast can cause persistent loss of ICMP PTB messages. Consider the example below:

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. 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 updates it PMTU estimate.

4.7. 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. Possible causes follow:

- Hardware inability to process fragmented packets.
- Failure to change vendor defaults.
- Unintentional misconfiguration.
- Intentional configuration (e.g., network operators consciously chooses to drop IPv6 fragments in order to address the issues raised in Section 4.1 through Section 4.6, 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:

- Manual configuration
- PMTUD
- PLPMTUD

Manual configuration is always applicable. If the MSS is configured to a sufficiently low value, the IP layer will never produce a packet whose length is greater than the protocol minimum link MTU. However, manual configuration prevents TCP from taking advantage of larger link MTU's.

Upper-layer protocols can implement PMTUD in order to discover and take advantage of larger path MTUs. However, as mentioned in Section 2.1, PMTUD relies upon the network 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 round-trip time, 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.ietf-tsvwg-datagram-plpmtud] proposes PLPMTUD procedures for DCCP and SCP.

Currently, User Data Protocol (UDP) [RFC0768] lacks a fragmentation mechanism of its own and relies on IP fragmentation. However, [I-D.ietf-tsvwg-udp-options] proposes a fragmentation mechanism for UDP.

5.2. Application Layer Solutions

[RFC8085] recognizes that IP fragmentation reduces the reliability of Internet communication. It also recognizes that UDP lacks a fragmentation mechanism of its own and relies on IP fragmentation. Therefore, [RFC8085] offers the following advice regarding applications the run over the UDP.

"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:

- DNS [RFC1035]
- OSPFv3 [RFC2328][RFC5340]
- Packet-in-packet 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 IP fragmentation may exist. They are not specifically considered in the context of this document.

6.1. DNS

DNS relies on UDP for efficiency, and the consequence is the use of IP fragmentation for large responses, as permitted by the DNS EDNS(0) options in the query. It is possible to mitigate the issue of fragmentation-based packet loss by having queries use smaller EDNS(0) UDP buffer sizes, or by having the DNS server limit the size of its UDP responses to some self-imposed maximum packet size that may be less than the preferred EDNS(0) UDP Buffer Size. In both cases, large responses are truncated in the DNS, signalling to the client to re-query using TCP to obtain the complete response. However, the operational issue of the partial level of support for DNS over TCP, particularly in the case where IPv6 transport is being used, becomes a limiting factor of the efficacy of this approach [Damas].

Larger DNS responses can normally be avoided by aggressively pruning the Additional section of DNS responses. One scenario where such pruning is ineffective is in the use of DNSSEC, where large key sizes act to increase the response size to certain DNS queries. There is no effective response to this situation within the DNS other than using smaller cryptographic keys and adoption of DNSSEC administrative practices that attempt to keep DNS response as short as possible.

6.2. OSPF

OSPF implementations can emit messages large enough to cause fragmentation. However, in order to optimize performance, most OSPF implementations restrict their maximum message size to a value that will not cause fragmentation.

6.3. Packet-in-Packet Encapsulations

In this document, packet-in-packet encapsulations include IP-in-IP [RFC2003], Generic Routing Encapsulation (GRE) [RFC2784], GRE-in-UDP [RFC8086] and Generic Packet Tunneling in IPv6 [RFC2473]. [RFC4459] describes fragmentation issues associated with all of the above-mentioned encapsulations.

The fragmentation strategy described for GRE in [RFC7588] has been deployed for all of the above-mentioned encapsulations. This strategy does not rely on IP 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.

See [I-D.ietf-intarea-tunnels] for further discussion.

7. Recommendations

7.1. For Application Developers

Application developers SHOULD NOT develop new applications that rely on IP fragmentation.

Application-layer protocols that depend upon IPv6 fragmentation SHOULD be updated to break that dependency. This can be achieved by using a sufficiently small MTU (e.g. The protocol minimum link MTU), disabling fragmentation, and ensuring that the transport protocol in use adapts its segment size to that MTU. This would avoid the problem of PMTUD failure described in Section 4.6. Another approach is to use PLPMTUD in a way suitable for the transport protocol in use (e.g. [I-D.ietf-tsvwg-datagram-plpmtud] for UDP).

7.2. For System Developers

Software libraries SHOULD include provision for PLPMTUD for each supported transport protocol.

7.3. For Middle Box Developers

Middle box developers SHOULD implement devices that support IP fragmentation. These boxes SHOULD not fail or cause failures when processing fragmented IP packets.

For example, in order to support IP fragmentation, a load balancer might execute the following procedure:

- Receive a fragmented packet
- Identify a next-hop using information drawn from the first fragment (i.e., the fragment containing offset 0)
- Forward the first fragment and all subsequent fragments through the above-mentioned next-hop

7.4. 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 4.6, filtering ICMPv6 PTB packets causes PMTUD to fail. Operators MUST ensure proper PMTUD operation in their network, including making sure the network generates PTB packets when dropping packets too large compared to outgoing interface MTU.

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 its use. 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

Thanks to Mikael Abrahamsson, Brian Carpenter, Silambu Chelvan, Lorenzo Colitti, Mike Heard, Tom Herbert, Tatuya Jinmei, Paolo Lucente, Manoj Nayak, Eric Nygren, and Joe Touch for their comments.

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

[Damas] Damas, J. and G. Huston, "Measuring ATR", April 2018, <http://www.potaroo.net/ispcol/2018-04/atr.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.ietf-intarea-tunnels] Touch, J. and M. Townsley, "IP Tunnels in the Internet Architecture", draft-ietf-intarea-tunnels-09 (work in progress), July 2018.

[I-D.ietf-tsvwg-datagram-plpmtud] Fairhurst, G., Jones, T., Tuexen, M., and I. Ruengeler, "Packetization Layer Path MTU Discovery for Datagram Transports", draft-ietf-tsvwg-datagram-plpmtud-05 (work in progress), October 2018.

[I-D.ietf-tsvwg-udp-options] Touch, J., "Transport Options for UDP", draft-ietf-tsvwg-udp-options-05 (work in progress), July 2018.

[Kent] Kent, C. and J. Mogul, ""Fragmentation Considered Harmful", In Proc. SIGCOMM '87 Workshop on Frontiers in Computer Communications Technology, DOI 10.1145/55483.55524", August 1987, <http://www.hpl.hp.com/techreports/Compaq-DEC/WRL-87-3.pdf>.

[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>.

[RFC2328] Moy, J., "OSPF Version 2", STD 54, RFC 2328, DOI 10.17487/RFC2328, April 1998, <https://www.rfc-editor.org/info/rfc2328>.

[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>.

[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>.

[RFC3128] Miller, I., "Protection Against a Variant of the Tiny Fragment Attack (RFC 1858)", RFC 3128, DOI 10.17487/RFC3128, June 2001, <https://www.rfc-editor.org/info/rfc3128>.

[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>.

[RFC4459] Savola, P., "MTU and Fragmentation Issues with In-the-Network Tunneling", RFC 4459, DOI 10.17487/RFC4459, April 2006, <https://www.rfc-editor.org/info/rfc4459>.

[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>.

[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>.

[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 Area WG
Internet-Draft
Intended status: Standard track
Expires March 4, 2019

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

August 31, 2018

Generic UDP Encapsulation
draft-ietf-intarea-gue-06

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/1id-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 March 4, 2019.

Copyright Notice

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

Abstract

This specification describes Generic UDP Encapsulation (GUE), which is a scheme for using UDP to encapsulate packets of different IP protocols for transport across layer 3 networks. By encapsulating packets in UDP, specialized capabilities in networking hardware for efficient handling of UDP packets can be leveraged. GUE specifies basic encapsulation methods upon which higher level constructs, such as tunnels and overlay networks for network virtualization, can be constructed. GUE is extensible by allowing optional data fields as part of the encapsulation, and is generic in that it can encapsulate packets of various IP protocols.

Table of Contents

1. Introduction

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

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

This document does not define any specific GUE extensions. [GUEEXTEN] specifies a set of initial 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

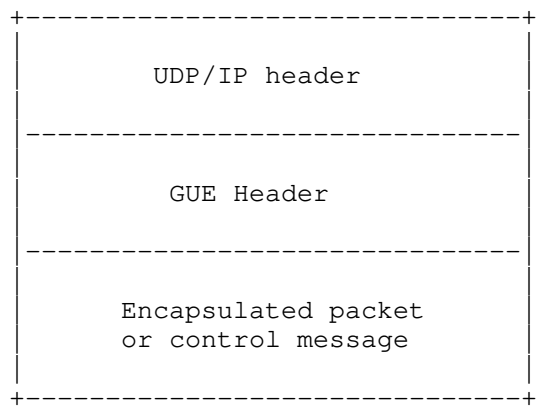
Control message	A formatted message in the GUE payload that is implicitly addressed to the decapsulator to monitor or control the state or behavior of a tunnel
Flags	A set of bit flags in the primary GUE header
Extension field	An optional field in a GUE header whose presence is indicated by corresponding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

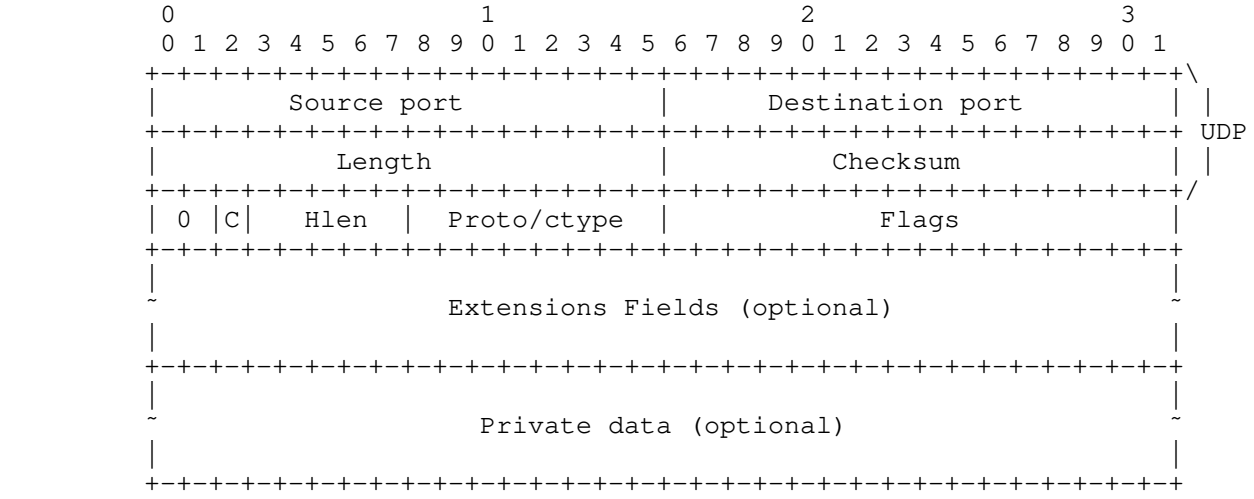
Variants 0 and 1 are described in this specification; variants 2 and 3 are reserved.

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:

- 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, the source port is either set to a flow entropy value as described in section 5.11, or it should be set to the GUE assigned port number, 6080.

- Destination port: If connection semantics (section 5.6.1) are applied to an encapsulation, this is set to the destination port for the tuple. If connection semantics are not applied this is set to the GUE assigned port number, 6080.

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

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

The GUE header consists of:

- Variant: 0 indicates GUE protocol version 0 with a header.
- C: C-bit: When set indicates a control message, not set indicates a data message.
- Hlen: Length in 32-bit words of the GUE header, including optional extension fields but not the first four bytes of the header. Computed as (header_len - 4) / 4, where header_len is the total header length in bytes. All GUE headers are a multiple of four bytes in length. Maximum header length is 128 bytes.
- Proto/ctype: When the C-bit is set, this field contains a control message type for the payload (section 3.2.2). When 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.
- Flags: Header flags that may be allocated for various purposes and may indicate presence of extension fields. Undefined header flag bits MUST be set to zero on transmission.
- Extension Fields: Optional fields whose presence is indicated by corresponding flags.
- Private data: Optional private data block (see section 3.4). If the private block is present, it immediately follows that last extension field present in the header. The private block is considered to be part of the GUE header. The length of this data is determined by subtracting the starting offset from the header length.

3.2. Proto/ctype field

The proto/ctype fields either contains an Internet protocol number (when the C-bit is not set) or GUE control message type (when the C-bit is set).

3.2.1 Proto field

When the C-bit is not set, the proto/ctype field MUST contain an IANA Internet Protocol Number. The protocol number is interpreted relative to the IP protocol that encapsulates the UDP packet (i.e. protocol of the outer IP header). The protocol number serves as an indication of the type of the next protocol header which is contained in the GUE payload at the offset indicated in Hlen. Intermediate devices MAY

parse the GUE payload per the number in the proto/ctype field, and header flags cannot affect the interpretation of the proto/ctype field.

When the outer IP protocol is IPv4, the proto field MUST be set to a valid IP protocol number usable with IPv4; it MUST NOT be set to a number for IPv6 extension headers or ICMPv6 options (number 58). An exception is that the destination options extension header using the PadN option MAY be used with IPv4 as described in section 3.6. The "no next header" protocol number (59) also MAY be used with IPv4 as described below.

When the outer IP protocol is IPv6, the proto field can be set to any defined protocol number except that it MUST NOT be set to Hop-by-hop options (number 0). If a received GUE packet in IPv6 contains a protocol number that is an extension header (e.g. Destination Options) then the extension header is processed after the GUE header is processed as though the GUE header is an extension header.

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

3.2.2 Ctype field

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

Control messages will be defined in an IANA registry. Control message types 1 through 127 may be defined in 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. Type 0 does not define a format of the control message. Instead, it indicates that the GUE payload is a control message, or part of a control message (as might be the case in GUE fragmentation), that cannot be correctly parsed or interpreted without additional context.

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 an initial 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.

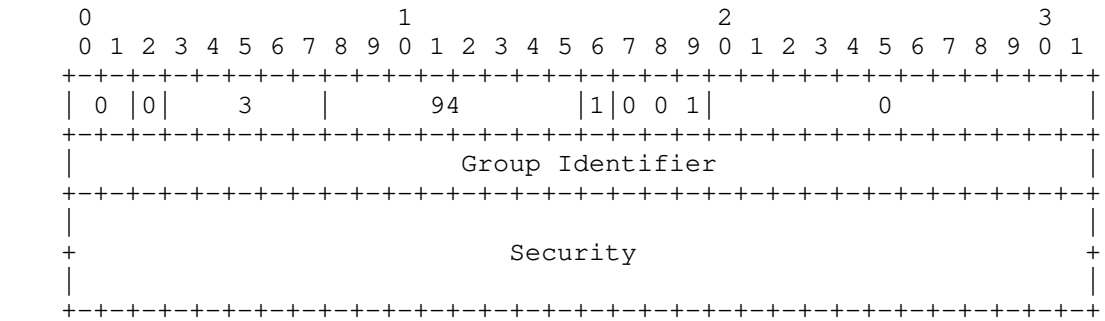
Extension fields are placed in order of the flags. New flags are to be allocated from high to low order bit contiguously without holes. Flags allow random access, for instance to inspect the field corresponding to the Nth flag bit, an implementation only considers the previous N-1 flags to determine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

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:

```
Private_length = (Hlen * 4) - Length(flags)
```

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

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

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

If a decapsulator receives a GUE packet with private data, it MUST validate the private data appropriately. If a decapsulator does not expect private data from an encapsulator, the packet MUST be dropped. If a decapsulator cannot validate the contents of private data per

the provided semantics, the packet MUST also be dropped. An implementation MAY place security data in GUE private data which if present MUST be verified for packet acceptance.

3.5. Message types

3.5.1. Control messages

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

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

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

3.5.2. Data messages

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

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

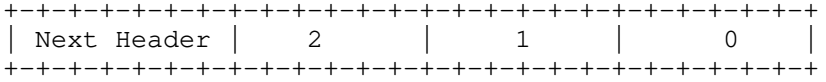
3.6. Hiding the transport layer protocol number

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:



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.

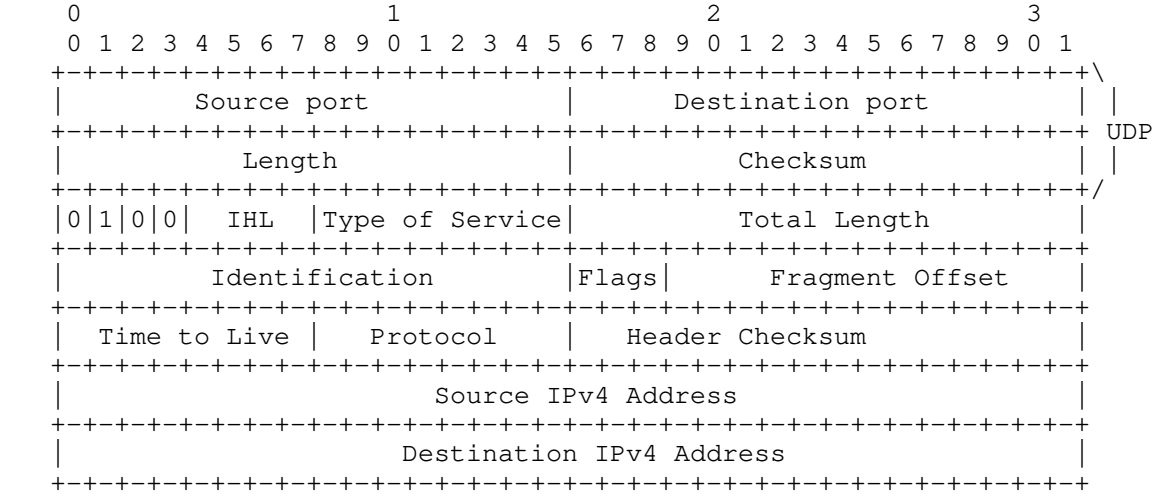
4. Variant 1

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:

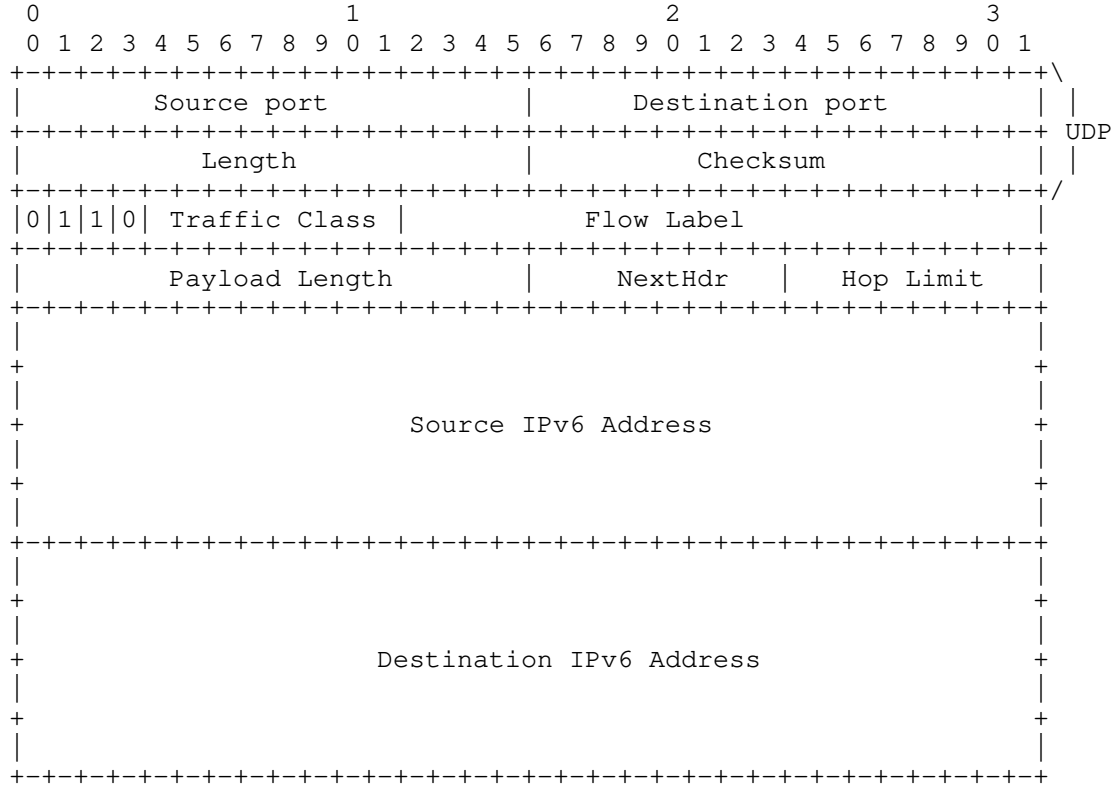


The UDP fields are set in a similar manner as described in section 3.1.

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:

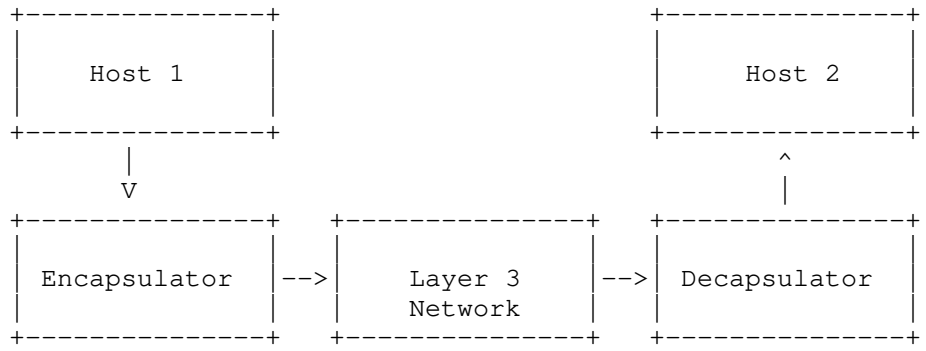


The UDP fields are set in a similar manner as described in section 3.1.

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; for example, the reverse path might not use GUE and/or any other form of encapsulation.



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 encapsualated in GUE then diffserv interaction [RFC2983] and ECN propagation for tunnels [RFC6040] SHOULD be followed.

5.4. Decapsulator operation

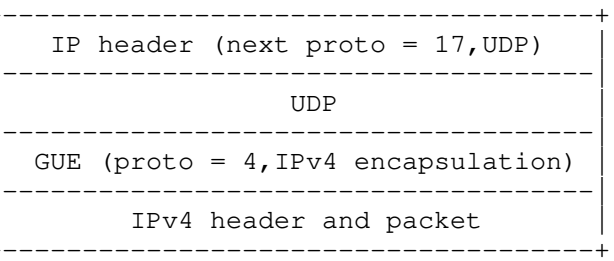
A decapsulator performs decapsulation of GUE packets. A decapsulator is addressed by the outer destination IP address of a GUE packet. The decapsulator validates packets, including fields of the GUE header.

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

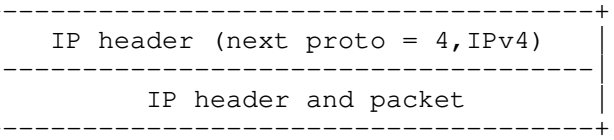
5.4.1. Processing a received data message

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 IPv4 packet using GUE variant 0. In this case proto field in the GUE header is set to 4 for IPv4 encapsulation:



The receiver removes the UDP and GUE headers and sets the next protocol field in the IP packet to 4, 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 IPv4 encapsulated 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 copy flags or fields to the outer GUE header. Each layer of encapsulation is considered independent.

5.6. Middlebox interactions

A middlebox 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 middlebox 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:

- UDP checksums are enabled (for IPv4 or IPv6).
- The GUE header checksum is used (defined in [GUEEXTEN]).
- 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, then 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:

- If the encapsulated packet is a layer 4 packet, TCP/IPv4 for instance, the flow entropy could be based on the canonical five-tuple hash of the inner packet.

- If the encapsulated packet is an AH transport mode packet with TCP as next header, the flow entropy could be a hash over a three-tuple: TCP protocol and TCP ports of the encapsulated packet.

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

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

5.11.2. Flow entropy properties

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

- The value set in the source port is within the ephemeral port range (49152 to 65535 [RFC6335]). Since the high order two bits of the port are set to one, this provides fourteen bits of entropy for the value.

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

- An encapsulator MAY occasionally change the flow entropy used for an inner flow per its discretion (for security, route selection, etc). To avoid thrashing or flapping the value, the

flow entropy used for a flow SHOULD NOT change more than once every thirty seconds (or a configurable value).

- Decapsulators, or any networking devices, SHOULD NOT attempt to interpret flow entropy as anything more than an opaque value. Neither should they attempt to reproduce the hash calculation used by an encapasulator in creating a flow entropy value. They MAY use the value to match further receive packets for steering decisions, but MUST NOT assume that the hash uniquely or permanently identifies a flow.

- Input to the flow entropy calculation is not restricted to ports and addresses; input could include flow label from an IPv6 packet, SPI from an ESP packet, or other flow related state in the encapsulator that is not necessarily conveyed in the packet.

- The assignment function for flow entropy SHOULD be randomly seeded to mitigate denial of service attacks. The seed 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:

- Configuration. The parameters used for a tunnel are configured at each endpoint.

- Negotiation. A tunnel negotiation can be performed. This could be accomplished in-band of GUE using control messages or private data.

- 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).

- 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 extensilbity, 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 GRE-in-UDP Encapsulation [RFC8086].

GUE has the following discriminating features:

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

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

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

- GUE is extensible. New flags and extension fields can be defined.

- 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.

- Private data in the encapsulation header allows local customization and experimentation while being compatible with processing in network nodes (routers and middleboxes).

- GUE includes both data messages (encapsulation of packets) and control messages (such as OAM).

- The flags-field model facilitates efficient implementation of extensibility in hardware. For instance, a TCAM can be use to parse a known set of N flags where the number of entries in the TCAM is 2^N. By comparison, the number of TCAM entries needed to parse a set of N arbitrarily ordered TLVS is approximately e*N!.

- GUE includes a variant that encapsulates IPv4 and IPv6 packets directly within UDP.

7. Security Considerations

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

- Authentication and integrity of the GUE header.
- 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	Control payload needs more context for interpretation	This document
1..127	Unassigned	
128..255	User defined	This document

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

[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. Priveleged ports

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

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

B.2. Setting flow entropy as a route selector

An encapsulator generating flow entropy in the UDP source port could modulate the value to perform a type of multipath source routing. Assuming that networking switches perform ECMP based on the flow hash, a sender can affect the path by altering the flow entropy. For instance, a host can store a flow hash in its protocol control block (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 protocols, such is GUE, are often supported in high speed network device hardware. Variable length header (VLH) protocols like GUE are often considered difficult to efficiently implement in hardware. In order to retain the important characteristics of an extensible and robust protocol, hardware vendors may practice "constrained flexibility". In this model, only certain combinations or protocol header parameterizations are implemented in hardware fast path. Each such parameterization is fixed length so that the particular instance can be optimized as a fixed length protocol. In the case of GUE this constitutes specific combinations of GUE flags, fields, and next protocol. The selected combinations would naturally be the most common cases which form the "fast path", and other combinations are assumed to take the "slow path".

In time, needs and requirements of the protocol may change which may manifest themselves as new parameterizations to be supported in the

fast path. To allow 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: March 4, 2019

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

August 31, 2018

Extensions for Generic UDP Encapsulation
draft-ietf-intarea-gue-extensions-05

Abstract

This specification defines a set of the initial 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/1id-abstracts.html

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

Copyright and License Notice

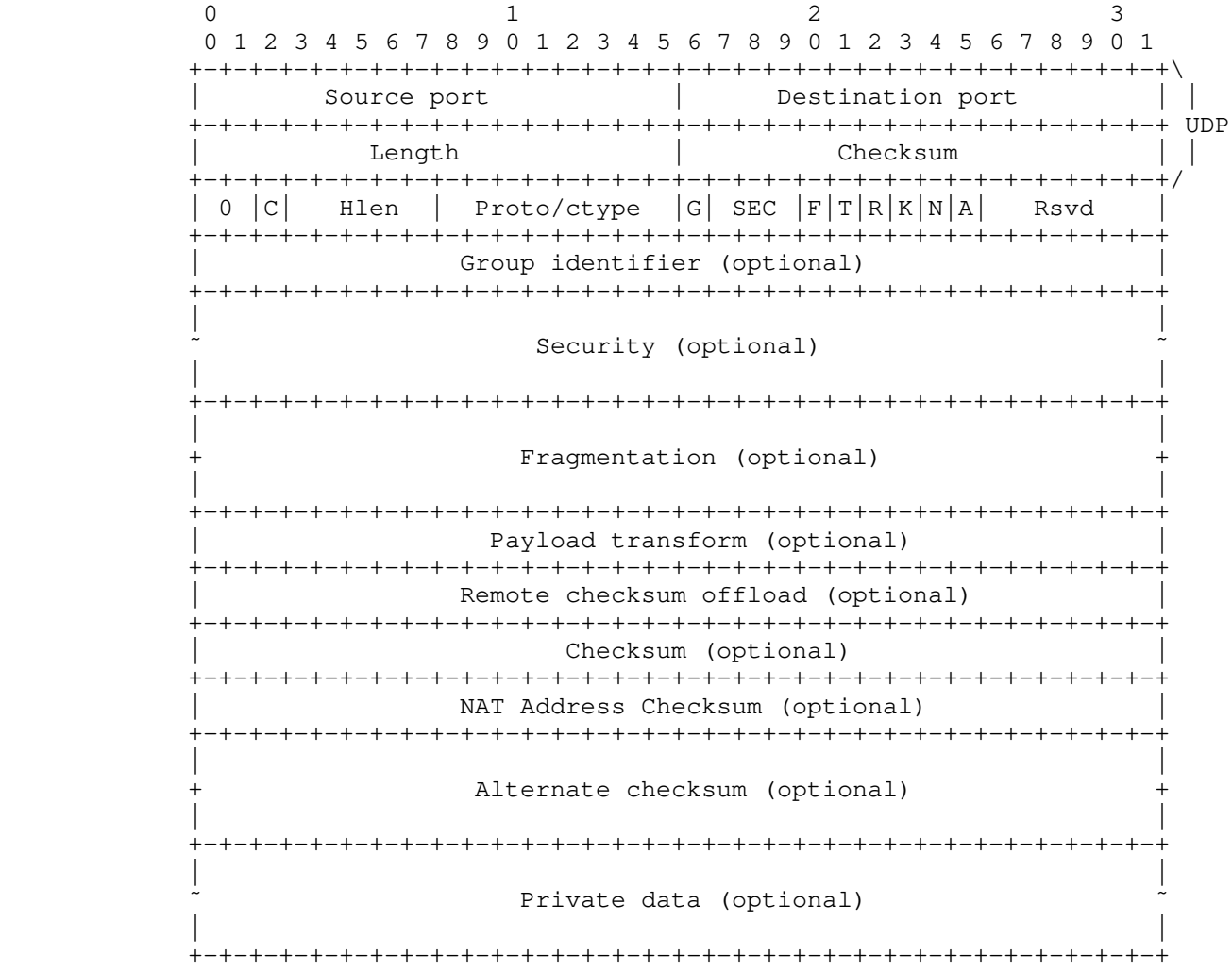
Table of Contents

1. Introduction

Generic UDP Encapsulation (GUE) [I.D.ietf-gue] is a generic and extensible encapsulation protocol. This specification defines an initial set of optional extensions for variant 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 variant 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:

- Variant: Set to 0 to indicate GUE encapsulation header. Note that variant 1 (direct IP encapsulation) does not allow options.

- C: C-bit. Indicates the GUE payload is a control message when set, a data message when not set. GUE optional extensions can be used with either control or data messages unless otherwise specified in the extension definition.

- 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 (header_len - 4) / 4. The length of the encapsulated packet is determined from the UDP length and the Hlen: encapsulated_packet_length = UDP_Length - 12 - 4 * Hlen.

- Proto/ctype: If the C-bit is not set this indicates IP protocol number for the packet in the payload; if the C bit is set this is the type of control message in the payload. The next header begins at the offset provided by Hlen. When the payload transform option or fragmentation option is used this field SHOULD be set to protocol number 59 for a data message, or zero for a control message, to indicate there is no parsable protocol in the payload.

- G: Indicates the the group identifier extension field is present. The group identifier option is described in section 3.

- SEC: Indicates security extension field is present. The security option is described in section 4.

- F: Indicates fragmentation extension field is present. The fragmentation option is described in section 5.

- T: Indicates payload transform extension field is present. The payload transform option is described in section 6.

- R: Indicates the remote checksum extension field is present. The remote checksum offload option is described in section 7.

- K: Indicates checksum extension field is present. The checksum option is described in section 8.

- N: Indicates NAT address checksum field is present. The NAT address checksum option is described in section 9.

- o A: 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].

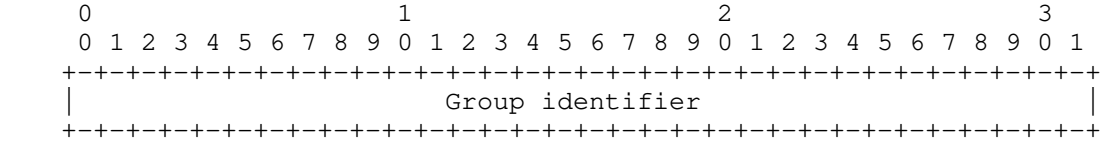
3. Group identifier option

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 by the G flag bit of the GUE header.

The format of the group identifier option is:



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.

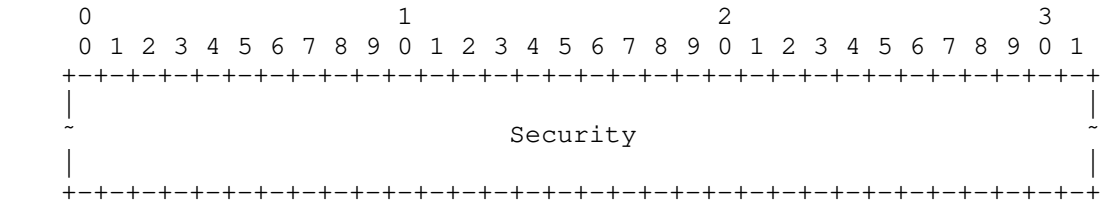
4. Security option

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 by the SEC flag bits of the GUE header.

The format of the security option is:



The fields of the option are:

- Security (variable length). Contains the security information. The specific semantics and format of this field 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:

- 000b - No security field
- 001b - 64 bit security field
- 010b - 128 bit security field
- 011b - 256 bit security field
- 100b - 320 bit security field (HMAC)
- 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.

If the GUE security option is present in packet, the receiver MUST validate the security before processing other fields or accepting the packet. If the security option is not present, but the encapsulator and decapsulator have agreed that security is required, the receiver MUST drop the packet as failing security checks. Note that this provision covers the case where the security flags bits are corrupted such that they are reset to zero which would be interpreted as no security field being present.

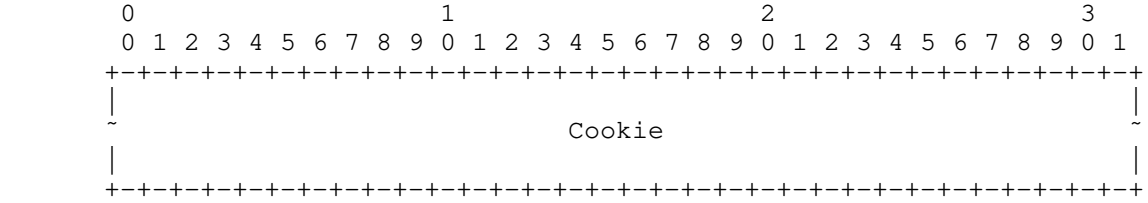
4.3. Cookies

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.

4.4.1. Extension field format

The cookie security option is a 64, 128, or 256 bit field. The security flags are set to 001b, 010b, 011b respectively for the corresponding field size.

The format of the field is:



Fields are:

- Cookie: Shared cookie value between encapsulator and decapsulator

4.3.1. Operation

4.3.1.1. Transmitter operation

The procedure for setting the GUE security cookie option on transmit

is:

1) Create the GUE header including the security field with the selected length for the cookie. Set the cookie to the value that is shared with decapsulator.

4.3.1.2. Receiver operation

The procedure for verifying the security cookie is:

1) Compare the received cookie to the expected shared cookie. If both the lengths are equal and the cookie values are equal then that packet is accepted, if the lengths or values are not equal then verification failed and the packet MUST be dropped.

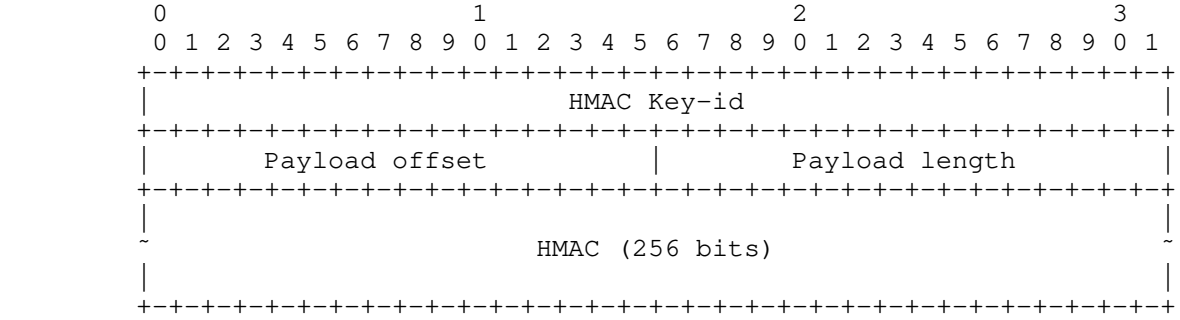
4.4. HMAC

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:

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

of payload data included in the HMAC.

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

- 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:

- The IP addresses

- The GUE header including all optional extensions and any private data. For the purposes of calculating the HMAC value, the HMAC value is set all zeroes.

- 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.

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]

4.4.4. Operation

4.4.4.1. Transmitter operation

The procedure for setting the GUE HMAC option on transmit is:

1) Create the GUE header including the 320 bit security field to hold the HMAC option. Set the HMAC Key-Id, payload length, and payload offset appropriately. The 16 byte HMAC field is initialized to zero.

2) Calculate the HMAC hash over the concatenation of the IP source and destination addresses. The particular hash and keys are agreed between the encpasulator and decapsulator out of band, and the key for input to the hash is the one indicated by the Key-Id amongst the set of shared keys.

3) Continue the the HMAC hash calculation 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). The calculation continues at the payload offset for payload length bytes.

5) Set the resultant hash value in the HMAC field.

4.4.4.2. Receiver operation

The procedure for verifying the HMAC security option is:

1) If the payload offset plus the payload coverage length is greater than the length of the encapsulated payload then drop the packet.

2) Note value in the HMAC field and set the HMAC field to zero.

3) Calculate the HMAC hash over the concatenation of the IP source and destination addresses. The particular hash and keys are agreed between the encpasulator and decapsulator out of band, and the key for input to the hash is the one indicated by the Key-Id amongst the set of shared keys.

4) Continue the the HMAC hash calculation from the start of the GUE header through the its length as indicated in GUE Hlen.

5) Continue the calculation to cover the payload portion if payload coverage is enabled (payload length field is non-zero). The calculation continues at the payload offset for payload length bytes.

6) Compare the computed HMAC value with the original value of the HMAC field. If they are equal then the packet is accepted, if they are not equal then verification failed and the packet MUST be dropped.

7) Restore the HMAC field to its original value.

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 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:

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

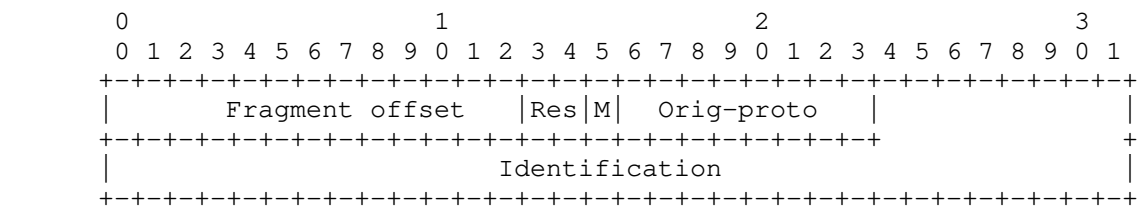
5.2. Scope

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.

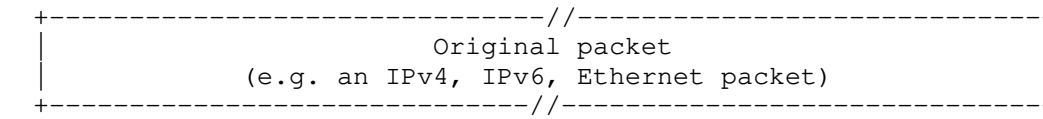
5.4. Fragmentation procedure

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).

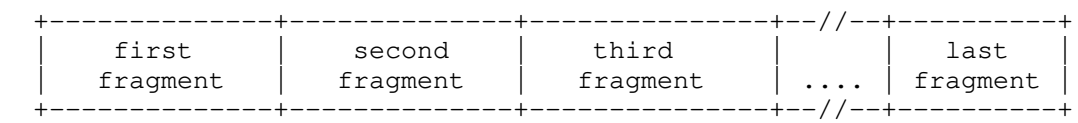
For every packet that is to be fragmented, the source node generates an Identification value. The Identification MUST be different than that of any other fragmented packet sent within the past 60 seconds (Maximum Segment Lifetime) with the same tunnel identification-- that is the same outer source and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

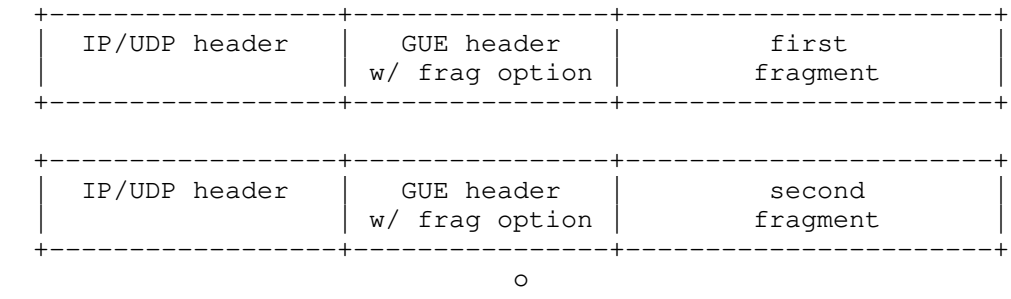


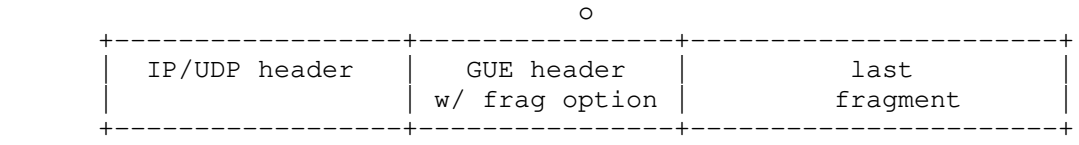
Fragmentation and encapsulation are performed on the original packet in sequence. First the packet is divided up in to fragments, and then each fragment is encapsulated. Each fragment, except possibly the last ("rightmost") one, is an integer multiple of 8 octets long. Fragments MUST be non-overlapping.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

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



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





Each fragment packet is composed of:

(1) Outer IP and UDP headers as defined for GUE encapsulation. The IP addresses and UDP ports MUST be the same for all fragments of a fragmented packet.

(2) A GUE header that indicates the fragmentation option is present. The C-bit and and proto/ctype are set appropriately as described above.

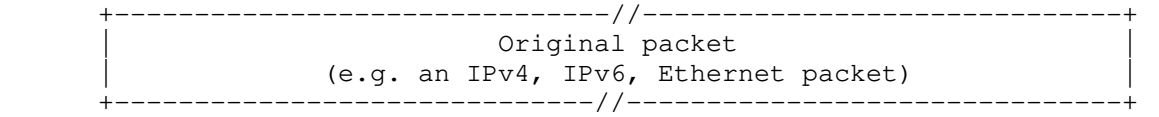
(3) The GUE fragmentation option. Orig-protocol is set to the protocol of the original packet. The M-bit is set for all fragments except the last one. Fragment offset is set as the offset of each fragment in the original packet.

(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:



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:

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

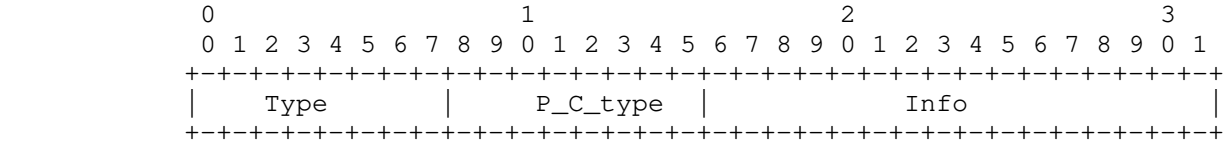
6. Payload transform option

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:



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 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 decaspsulator MUST decrypt the payload before accepting the packet. If a decapsulator fails to perform the reverse transformation or cannot validate the transformation it MUST discard the packet and MAY generate an alert

to the management system.

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.

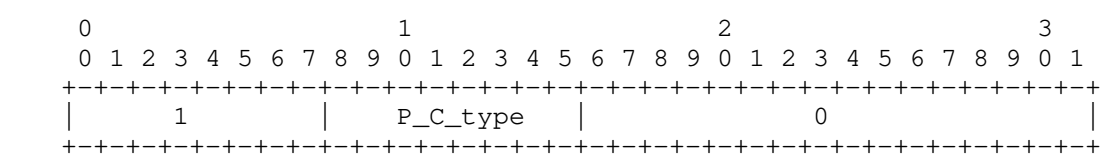
If both the security option and the payload transform are used in a GUE packet, an encapsulator MUST perform the payload transformation first, set the payload transform option in the GUE header, and then create the security option. A decapsulator does processing in reverse-- the security option is processed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



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

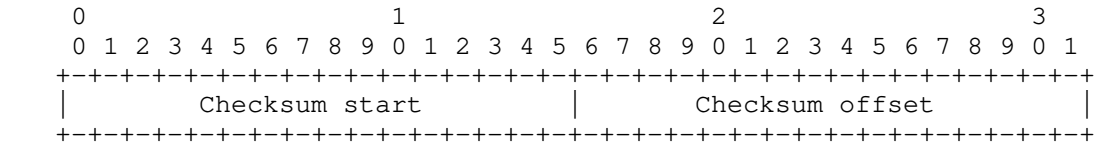
Remote checksum offload is mechanism that provides checksum offload of encapsulated packets using rudimentary offload capabilities found in most Network Interface Card (NIC) devices. Many NIC implementations can only offload the outer UDP checksum in UDP encapsulation. Remote checksum offload is described 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 for an inner encapsulated packet. Effectively this offloads the computation of the inner checksum. Enabling the outer checksum in encapsulation has the additional advantage that it covers more of the packet, including the encapsulation headers, than an inner checksum.

7.1. Extension field format

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

The format of remote checksum offload field is:



The fields of the option are:

- Checksum start: starting offset for checksum computation relative to the start of the encapsulated payload. This is typically the offset of a transport header (e.g. UDP or TCP).

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

7.2. Usage

7.2.1. Transmitter operation

The typical actions to set remote checksum offload on transmit are:

1) Transport layer creates a packet and indicates in internal packet meta data that checksum is to be offloaded to the NIC (normal transport layer processing for checksum offload). The checksum field is populated with the bitwise not of the checksum of the pseudo header or zero as appropriate.

2) Encapsulation layer adds its headers to the packet including the remote checksum offload option. The start offset and checksum offset are set accordingly.

3) Encapsulation layer arranges for checksum offload of the outer header checksum (e.g. UDP).

4) Packet is sent to the NIC. The NIC will perform transmit checksum offload and set the checksum field in the outer header. The inner header and rest of the packet are transmitted without modification.

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 then the length of the packet minus two, then the packet MUST be dropped.

3) Deduce full checksum for the IP packet. If a NIC is capable of receive checksum offload it will return either the full checksum of the received packet or an indication that the UDP checksum is correct. Either of these methods can be used to deduce the checksum over the IP packet [UDPENCAP].

4) From the packet checksum, subtract the checksum computed from the start of the packet (outer IP header) to the offset in the packet indicted by checksum start in the meta data. The result is the deduced checksum to set in the checksum field of the encapsulated transport packet.

   In pseudo code:

```
csum: initialized to checksum computed from start (outer IP
      header) to the end of the packet
start_of_packet: address of start of packet
```

```
encap_payload_offset: relative to start_of_packet
csum_start: value from 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.

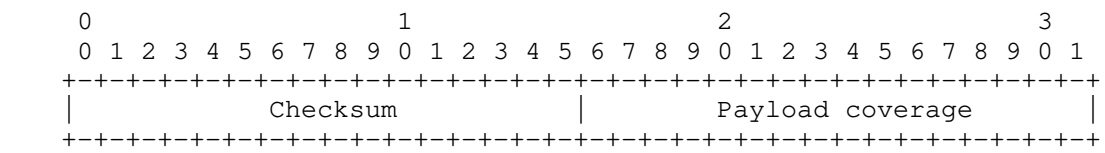
8. Checksum option

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:



The fields of the option are:

- Checksum: Computed checksum value. This checksum covers the GUE header (including fields and private data covered by Hlen), the GUE pseudo header, and optionally all or part of the payload (encapsulated packet).

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

8.2. Requirements

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

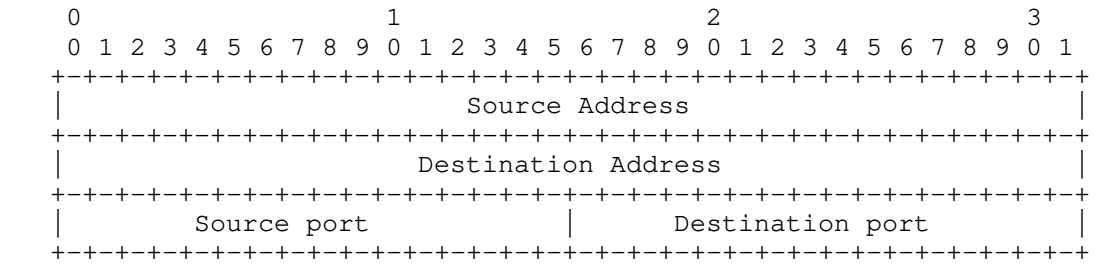
The GUE header checksum SHOULD be used when the UDP checksum is zero for IPv4 if the GUE header includes data that when corrupted can lead to misdelivery or other serious consequences, and there is no other mechanism that provides protection (no security field that checks integrity for instance).

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).

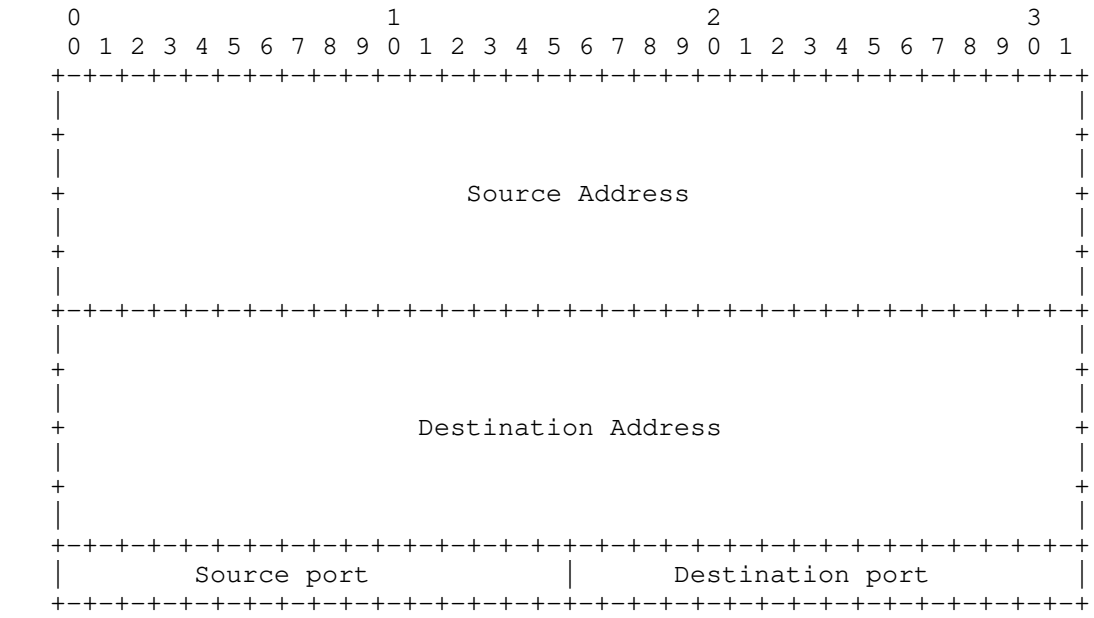
8.3. GUE checksum pseudo header

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 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 GUE pseudo header for IPv4 or IPv6.

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 checksum flag

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 depcapsulator might agree that checksum is always required. This agreement could be established by configuration or 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.

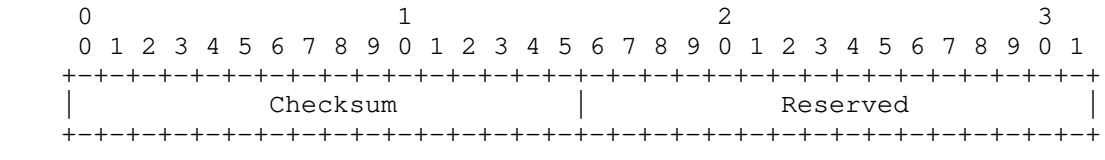
9. NAT checksum address option

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. Transmitter operation

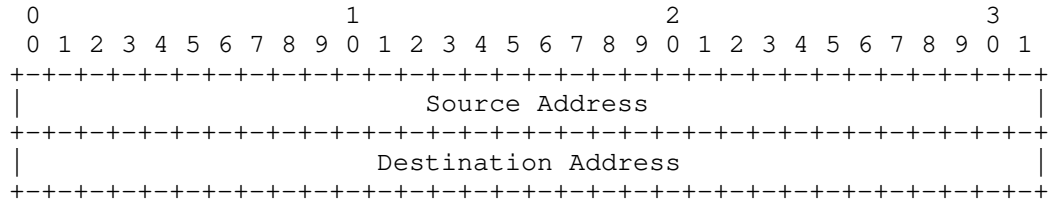
The procedure for setting the GUE checksum on transmit is:

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

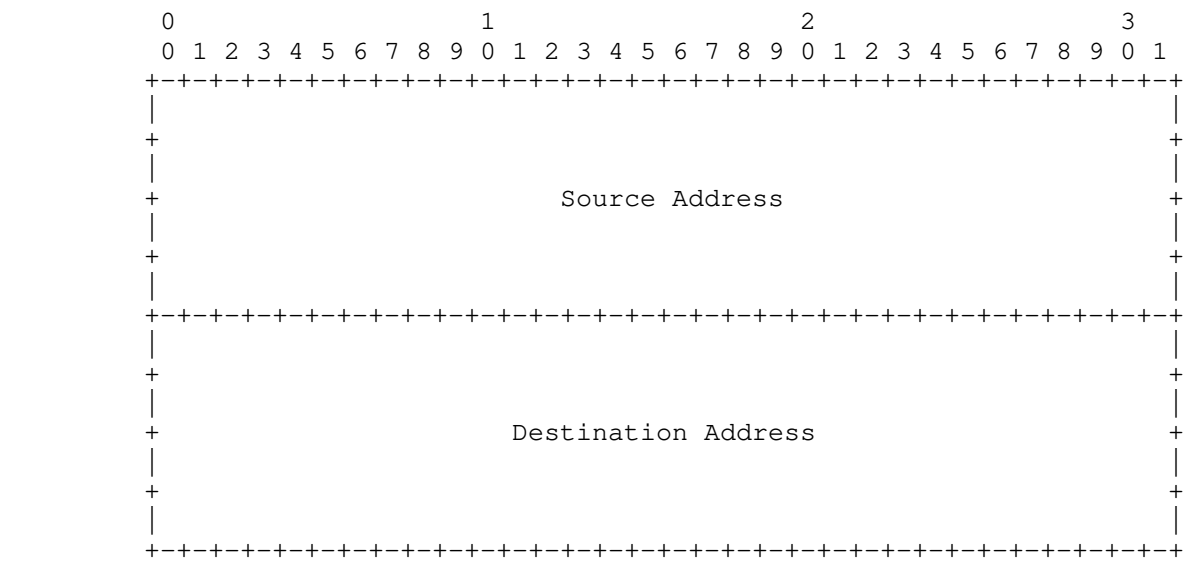
1) Compute the ones complement checksum over the source and destination IPv4 or IPv6 addresses.

2) Set the resultant value in the Checksum field.

For IPv4 the checksum is computed over:



For IPv6 the checksum is computed over:



9.2.2. Receiver operation

1) Validate the UDP checksum is correct

2) Compute the checksum over the IP addresses 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
```

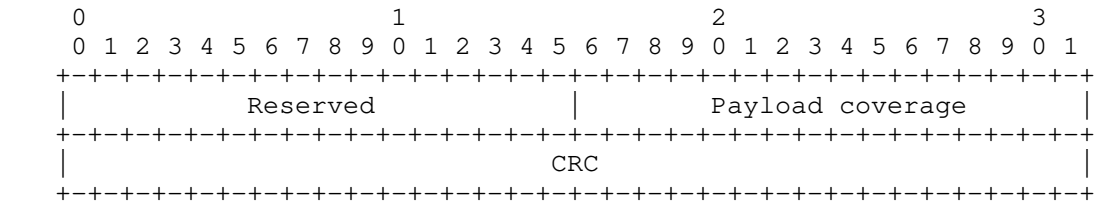
10. Alternative checksum option

The alternative checksum option contains a check over the GUE header and optionally all or part of the GUE payload. The algorithm used is CRC-32C which is the same as that used by iSCSI and SCTP. The alternative checksum can provide stronger protection against data corruption than that provided by the one's complement checksum used in the UDP checksum or the GUE checksum (section 8).

10.1. Extension field format

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

The format of alternate checksum field is:



The fields of the option are:

- 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).

- 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.

10.2. Usage

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

CRC-32C is calculated using the 0x1EDC6F41 polynomial:

```
x^32 + x^28 + x^27 + x^26 + x^25 + x^23 + x^22 + x^20 + x^19 +
x^18 + x^14 +x^13 + +x^11 + x^10 +x^9 +x^8 + x + 1
```

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 CRC using the CRC-32C algorithm 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 to four bytes, 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 CRC using the CRC-32C algorithm 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 to four bytes, 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 alternate checksum flag

Similar to the GUE checksum, the GUE alternate checksum does not protect against the alternative checksum flag (A flag) being corrupted. If an encapsulator sets the alternative checksum flags and option but the A bit 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 depcapsulator 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, or 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 alternate checksum and set the alternate checksum option.

8) Calculate GUE checksum and set checksum option.

11.2. Processing order when receiving

On reception the order of actions is:

1) Verify GUE checksum.

2) Verify alternate checksum. If the GUE checksum option is present, set its checksum fields to zero for computing the alternate checksum. After computation, restore the checksum value in the GUE 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 between GUE extensions and private fields is unspecified in this specification. Any processing order requirements regarding private data must be agreed upon between an ecapsulator and decapsulator.

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 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 a GUE transform for encryption 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 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. This document requests an initial assignment of flags in the registry.

IANA is requested to assign flags for the extensions defined in this specification. Specifically, an assignment is requested for the Group Identfier, 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	4 bytes	Alternate checksum	This document
Bit 10..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: April 22, 2019

P. Pfister
E. Vyncke, Ed.
Cisco
T. Pauly
D. Schinazi
Apple
W. Shao
Telecom-ParisTech
October 19, 2018

Discovering Provisioning Domain Names and Data
draft-ietf-intarea-provisioning-domains-03

Abstract

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

This document describes a way for hosts to identify such contexts, called Provisioning Domains (PvDs), where Fully Qualified Domain Names (FQDNs) act as PvD identifiers. 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 April 22, 2019.

Table of Contents

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 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. See also [RFC7556].

PvD-aware host A host that supports the association of network configuration information into PvDs and the use of these PvDs. Also named PvD-aware node in [RFC7556].

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); else, the PvD ID MUST be different to follow section 2.4 of [RFC7556].

3.1. PvD ID Option for Router Advertisements

This document introduces a Router Advertisement (RA) option called PvD 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 3.4.2), 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.

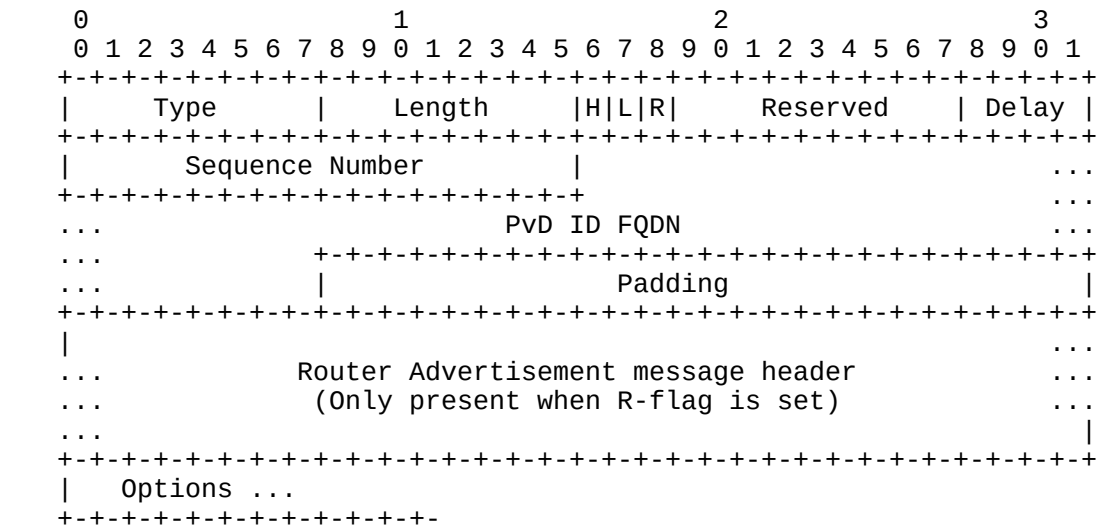


Figure 1: PvD ID Router Advertisements Option format

Type : (8 bits) Set to 21.

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 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 3.4.2).

R-flag : (1 bit) 'Router Advertisement' flag stating whether the PvD Option is followed (right after padding to the next 64 bits boundary) by a Router Advertisement message header (See section 4.2 of [RFC4861]).

Delay : (4 bits) Unsigned integer used to delay HTTP GET queries from hosts by a randomized backoff (see Section 4.1).

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 R-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 Option such as to be ignored by hosts that are not 'PvD-aware'.

Here is an example of a PvD option with example.org as the PvD ID FQDN and including a RDNSS and prefix information options (it also have the sequence number 123, presence of additional information to be fetched with a delay indicated as 5):

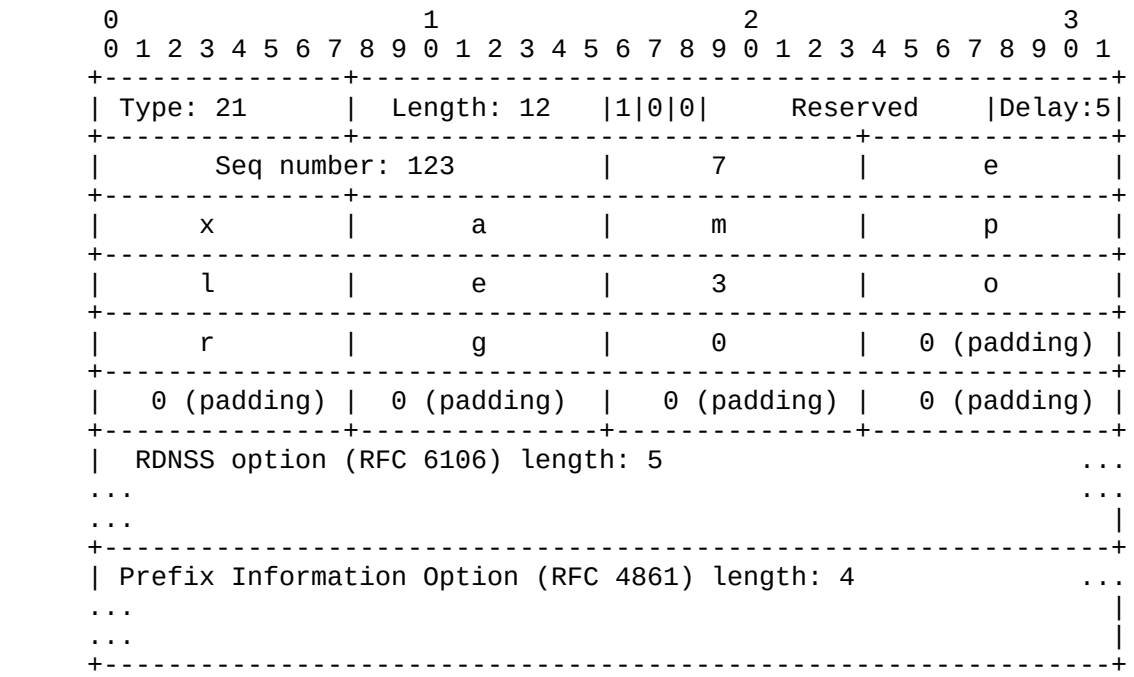


Figure 2

3.2. Router Behavior

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

The PvD 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 by different RAs, MUST use different link-local addresses because these implicit PvDs are identified by the source addresses of the RAs.

As specified in [RFC4861], when the set of options causes the size of an advertisement to exceed the link MTU, multiple router advertisements can be sent, each containing a subset of the options. In such cases, the PvD 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 options.

3.3. Non-PvD-aware Host Behavior

As the PvD Option has a new option code, non-PvD-aware hosts will simply ignore the PvD Option and all the options it contains. This ensure the backward compatibility required in section 3.3 of [RFC7556]. This behavior allows for a mixed-mode network with a mix of PvD-aware and non-PvD-aware hosts coexist.

3.4. PvD-aware 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 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 options are found in a given RA, hosts MUST ignore all but the first PvD 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 when forwarding packets ([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].

3.4.1. 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).

3.4.2. 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 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.

3.4.3. Connection Sharing by the Host

The situation when a host shares connectivity from an upstream interface (e.g. cellular) to a downstream interface (e.g. WiFi) is known as 'tethering'. Techniques such as ND-proxy [RFC4389], 64share [RFC7278] or prefix delegation (e.g. using DHCPv6-PD [RFC3633]) may be used for that purpose.

Whenever the RAs received from the upstream interface contain a PVD RA option, hosts that are sharing connectivity SHOULD include a PVD Option within the RAs sent downstream with:

The same PVD-ID FQDN.

The same H-bit, Delay and Sequence Number values.

The L bit set whenever the host is sharing IPv4 connectivity received from the same upstream interface.

The bits from the Reserved field set to 0.

The values of the R-bit, Router Advertisement message header and Options field depend on whether the connectivity should be shared only with PvD aware hosts or not (see Section 3.2). In particular, all options received within the upstream PvD option and included in the downstream RA SHOULD be included in the downstream PvD option.

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 3.4.

4.1. Retrieving the PvD Additional Information

When the H-flag of the PvD 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 3.4. 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 host has

a temporary address per [RFC4941] in this PvD, 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, object updates are delayed by a randomized backoff time.

When a host performs an object update after it detected a change in the PvD Option Sequence number, it MUST delay the query by a random time between zero and 2**(Delay * 2) milliseconds, where 'Delay' corresponds to the 4 bits long unsigned integer in the last received PvD Option.

When a host last retrieved an object at time A including a validity time B, and is configured to keep the object up to date, it MUST perform the update at a uniformly random time in the interval [(B-A)/2,B].

In the example Figure 2, the delay field value is 5, this means that host MUST delay the query by a random number between 0 and 2**(5 * 2) milliseconds, i.e., between 0 and 1024 milliseconds.

Since the ’Delay’ value is directly within the PvD Option rather than the object itself, an operator may perform a push-based update by incrementing the Sequence value while changing the Delay value depending on the criticality of the update and its PvD Additional Information servers capacity.

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 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 Option 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 IPv6 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.

Note that this check cannot be performed when the HTTPS query is performed over IPv4. Therefore, the PvD ID FQDN SHOULD NOT have a DNS A record whenever all hosts using the given PvD have IPv6 connectivity.

5. Operational Considerations

This section describes some use cases of PvD. For the 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 option that also contains other options will be described as:

- RA Header: router lifetime = 6000
- Prefix Information Option: length = 4, prefix = 2001:db8:cafe::/64
- PvD Option header: length = 3+ 5 +4 , PvD ID FQDN = example.org., R-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 mixed environment of PvD-aware hosts and non-PvD-aware 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 non-PvD-aware hosts:

- RA Header: router lifetime = 6000 (non-PvD-aware hosts will use this router as a default router)
- Prefix Information Option: length = 4, prefix = 2001:db8:cafe::/64
- Recursive DNS Server Option: length = 3, addresses= [2001:db8:cafe::53]
- PvD Option header: length = 3+ 2, PvD ID FQDN = foo.example.org., R-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:

- RA Header: router lifetime = 0 (non-PvD-aware hosts will not use this router as a default router)

- PvD Option header: length = 3+ 2 + 4 + 3, PvD ID FQDN = example.org., R-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, non-PvD-aware 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 adress.

6. Security Considerations

Although some solutions such as IPsec or SeND [RFC3971] can be used in order to secure the IPv6 Neighbor Discovery Protocol, in practice 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 advertize 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 connection 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

Upon publication of this document, IANA is asked to remove the 'reclaimable' tag off the value 21 for the PvD option (from the IPv6 Neighbor Discovery Option Formats registry).

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 option Flags" reserving bit positions from 0 to 15 to be used in the PvD 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, Zhen Cao, Tim Chow, Lorenzo Colitti, Ian Farrer, Bob Hinden, Tatuya Jinmei, Erik Kline, Ted Lemon, Jen Lenkova, Veronika McKillop, 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>.

[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>.

[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>.

[RFC4389] Thaler, D., Talwar, M., and C. Patel, "Neighbor Discovery Proxies (ND Proxy)", RFC 4389, DOI 10.17487/RFC4389, April 2006, <https://www.rfc-editor.org/info/rfc4389>.

[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>.

[RFC6731] Savolainen, T., Kato, J., and T. Lemon, "Improved Recursive DNS Server Selection for Multi-Interfaced Nodes", RFC 6731, DOI 10.17487/RFC6731, December 2012, <https://www.rfc-editor.org/info/rfc6731>.

[RFC7278] Byrne, C., Drown, D., and A. Vizdal, "Extending an IPv6 /64 Prefix from a Third Generation Partnership Project (3GPP) Mobile Interface to a LAN Link", RFC 7278, DOI 10.17487/RFC7278, June 2014, <https://www.rfc-editor.org/info/rfc7278>.

[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.

A.6. WG Document version 02

Use the IANA value of 21 in the text and update the IANA considerations section accordingly

add the Delay field to avoid the thundering herd effect

add Wenqin Shao as author

keep the 1 PvD per RA model

changed the intro (per Zhen Cao) "when choosing which PvD and transport should be used" => "when choosing which PvD should be used"

rename A-flag in R-flag to avoid A-flag of PIO

use the wording "PvD Option", removing the ID token as it is now a container with more then just an ID, removing ’RA’ in the option name to be consistent with other IANA NDP option

use "non-PvD-aware" rather than "PvD-ignorant"

added more reference to RFC 7556 (notably for PvD being globally unique, introducing PvD-aware host vs. PvD-aware node)

Section 3.4.3 renamed from "interconnection shared by node" to ’connection shared by node"

Section 3.4 renamed into "PvD-aware Host Behavior"

Added a section "Non-PvD-aware Host Behavior"

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

Wenqin Shao
Telecom-ParisTech
France

Email: wenqin.shao@telecom-paristech.fr

TSVWG
INTERNET-DRAFT
Intended Status: Informational
Expires: April 12, 2019

Y. Li
X. Zhou
Huawei

October 9, 2018

Overlayed Path Segment Forwarding (OPSF) Problem Statement
draft-li-tsvwg-overlayed-path-segment-fwding-ps-00

Abstract

Various overlays are used in networks including WAN, enterprise campus and others. End to end path are divided into multiple segments some of which are overlay encapsulated to achieve better path selection, lower latency and so on. Traditional end-to-end transport layer is not very responding to microburst and non-congestive packet loss caused by the different characteristics of the path segments. With the potential transport enhancement for the existing or purposely created overlayed path segment, end to end throughput can be improved. This document illustrates the problems in some use cases and tries to inspire more about whether and how to solve them by introducing a reliable, efficient and non-intrusive transport forwarding over the overlayed path segment(s).

Table of Contents

1. Introduction

Overlay tunnels are widely deployed for various networks, including long haul WAN interconnection, enterprise wireless access networks, etc. End to end connection are normally broken into multiple path segments for different purposes, for instance, selecting a better overlay path over the WAN or deliver the packets over the heterogenous networks like enterprise access and core networks.

TCP-like transport layer provides end to end flow control and congestion control for path reliability and high throughput. Such an approach has the problems of slow congestion responding and non-congestive loss misinterpretation at the sender and does not achieve the optimal performance in certain cases.

Some of the problems have been well known over years. With new technologies are emerging like NFV (Network Function Virtualization) and various flexible overlay protocols, forwarding over the specific overlayed path segment(s) can be considered to be enhanced by providing a reliable and non-intrusive transport to improve the throughput to solve those problems.

This document illustrates the problems in some use cases and tries to inspire more about whether and how to solve them by introducing a reliable, efficient and non-intrusive transport forwarding for the overlayed path segment(s).

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

OPSF: Overlayed Path Segment Forwarding

3. Use Cases and Problems

The following subsections presents use cases from different scenarios using overlay tunnels with a common need of higher performance and reliable overlayed path segment in best effort networks.

3.1 Microburst in Long Haul Network

Internet is a huge sized network of networks. The interconnections of end devices using this global network are normally provided by ISPs (Internet Service Provider). This ISP provided huge network is considered as traditional Internet. CSPs (Cloud Service Provider) are connecting their data centers using Internet or self-constructed networks/links. This expands Internet's infrastructure and together with the original ISP's infrastructure, forms the Internet underlay.

NFV further makes it easier to dynamically provision a new virtual node as a work load in a cloud for CPU/storage intensive functions. With the aid of various mechanisms such as kernel bypassing and Virtual IO, forwarding based on virtual node is becoming more and more effective. The interconnections among the purposely positioned virtual nodes and/or the existing nodes with vitalization functions potentially form "the Second Plane" or overlay of Internet. It is called the Cloud-Internet Overlay Network (CION) in this document.

CION makes use of overlay technologies to direct the traffic going to the specific path regardless the underlying physical topology to achieve better service delivery. Figure 1 shows an emerging multi-segment overlay over large geographic distances. It purposely creates or selects overlay nodes (ON) from clouds/Internet. Segment here is the virtual hop between two ONs. By directing the traffic to be forwarded along those virtual nodes rather than the default path, better delivery in terms of throughput and delay can be achieved. When a large number of potential virtual nodes are available, there is a high chance that a better path could be found [CRONets].

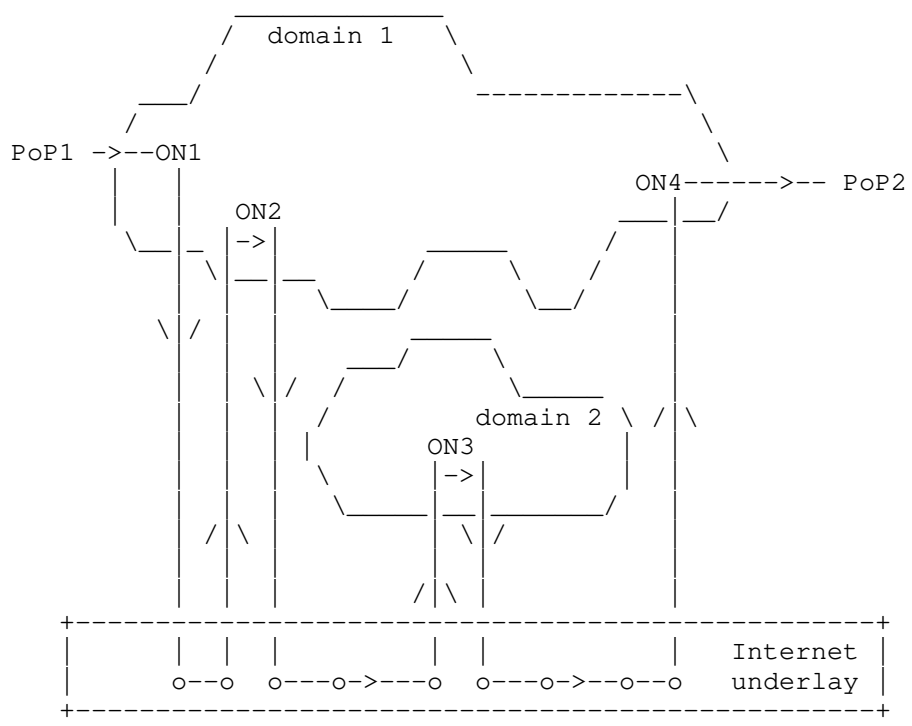


Figure 1. Cloud-Internet Overlay Network (CION)

Microburst is an unexpected data bursts within a very small time window (probably in micro-seconds). Some research shows microbursts happen even for underutilized link [BurstyAna]. The short spikes caused by microburst result in higher jitter and sometimes packet loss in a network. Such loss may trigger the congestion control like reducing the sending rate at the TCP sender as it exhibits the normal pattern of congestion loss in terms of duplicate acknowledgements and/or RTT increases. As microburst is extremely short, the packet loss caused by it is non-persistent and rather random. Therefore it does not necessarily require the sender to reduce its sending rate. Invoking the congestion control at the sender may unnecessarily make the average sending rate low and degrades the throughput in long haul CION. In addition, long haul transmission may take hundreds of milliseconds. The packet loss response at the sender to microburst over the long haul transmission is not timely. Sender's reaction does not really respond to the current instantaneous path situation.

Overlay nodes in the middle can potentially offer new possibilities,

e.g. retransmission over ONs, to better response to microbursts. Such enhancement can be enabled based on the individual overlayed path segment rather than on the entire end to end path to improve the response time and performance from the packet loss/re-order caused by microburst. Such enhancement should avoid racing with higher layer transport protocols.

3.2 Non-congestive Loss in WiFi Accessed Campus Overlay

Different path segments have different characteristics. The probabilities of packet loss over every and each segments have a large variance. The non-congestive packet loss usually occurs in some specific overlayed path segments. End to end TCP-like transport protocols do not take this factor into careful account. It assumes that packet loss for any reason is almost evenly distributed across the entire path, and adjusts the sender to accommodate the packet loss of the bottleneck segment. This results in non-optimal sending rate in some cases.

Figure 2 shows the WiFi accessed enterprise campus. AP connects to its edge switch normally using Cat5/5e twisted-pair cable which typically provides less than 10G bandwidth. The data packets are tunneled using various overlay mechanisms, like VXLAN [RFC7348], LISP [RFC6830] or CAPWAP [RFC5415]. Two edge switches use another overlay segment over campus core network to deliver the packets which provides more functions like policy enforcement and mobility enhancement. This overlay is usually over fiber which provides higher bandwidth.

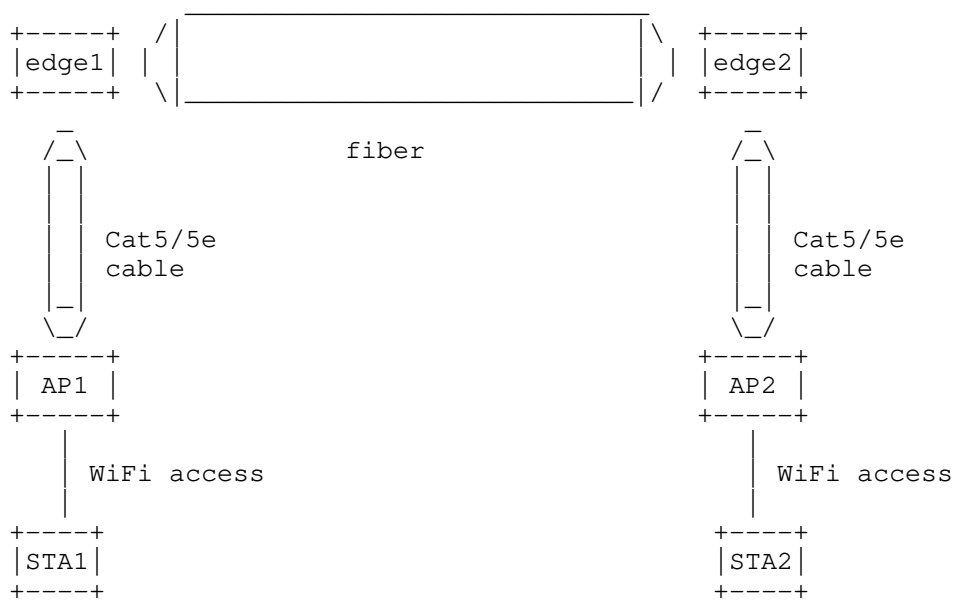


Figure 2. WiFi accessed Campus Overlay Network

Cat5/5e cables, especially UTP (Unshielded twisted pair), are susceptible to distance, interference, and bundling. The environment and the way they are deployed cause drastic changes in random loss rate. The overlay tunnel running over it will have more transmission unreliability than the overlay running on the fiber. Current transport layer is not able to identify such specific problematic segment and simply leaves it for the end to end congestion control to handle it so that the sender may be kept at a lower sending rate and the throughput is not optimal.

In addition to the uplink of the AP, the non-congestive packet loss generated by the wireless access link itself accounts for the largest proportion in the end-to-end path. Wifi access is affected by fades, interference, attenuation and corruption. Non-congestive loss is common. Its link layer has mechanisms to do the packet recovery. However the number of local link layer retransmission is usually based on the empirical value or the static configuration. When the value is not properly chosen, the TCP sender can be unnecessarily exposed by the random packet loss and reduce the sending rate. It is hard to make the link layer frame recovery work in concert with the current end to end transport layer.

3.3 Higher Reliability and Low Latency for Interactive Application

Mobile gaming and VoIP like application normally can not tolerate a retransmission even over a path segment. When two divergent overlay segments are available like shown in figure 3 for path from ON1 to ON2, purposely duplicating packets over two segments provides more reliability. Two disjoint segments can usually be obtained by measuring to find segments with very low mathematical correlation in latency change. When the number of overlay nodes is large, it is easy to find such disjoint segments. Random node or memory failure may also benefit from the duplicating packets over disjoint segments.

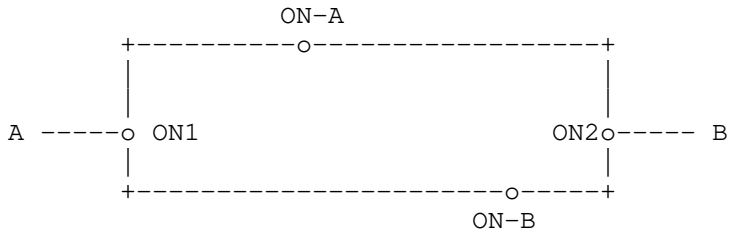


Figure 3. Multiple Overlayed Path Segments for Higher Reliability

4. Features to be Considered for OPSF (Overlayed Path Segment Forwarding)

The diagram shown in Figure 4 illustrates a typical scenario with an overlayed path segment. Transport layer provide the end to end flow control between two end host. When an overlayed path segment exists or is purposely created between two overlay nodes, an enhanced forwarding over that segment can potentially solve some problems of end to end transport performance issues and at the same time provides more reliability and flexibility to traffic path.

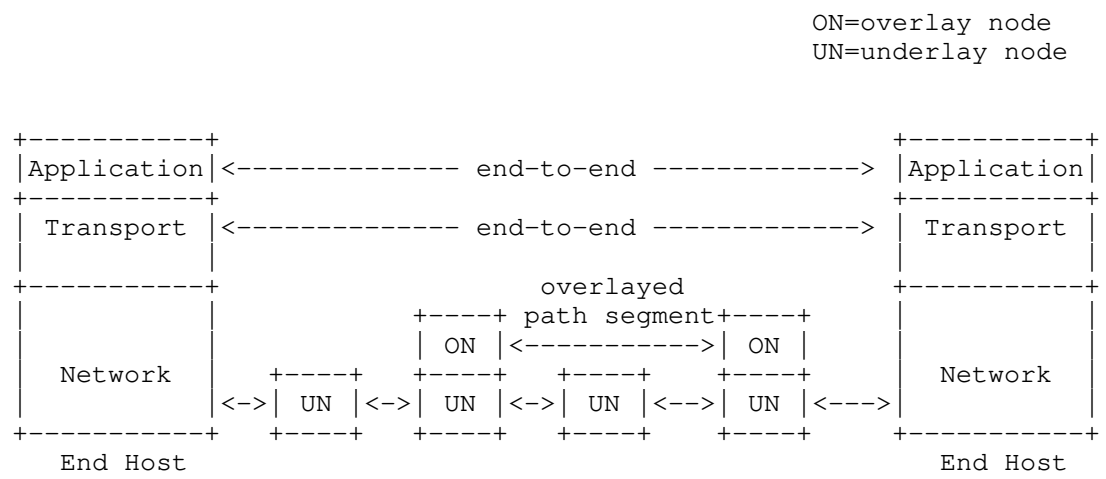


Figure 4. A Simple Overlayed Path Segment Forwarding Usage Scenario

Features need more investigations include,

- Enhancement for the overlayed path segment forwarding/transport, like retransmission, FEC(forward error correction), duplicating packet over the segments, lightweighted congestion control, etc. When the segment is a small portion of the whole end to end path, the retransmission over it has more benefit. Retransmission over the path segment has to be carefully designed to avoid the racing condition with the upper layer. The segment enabled retransmission may measure the segment RTT by itself to determine the appropriate retransmission attempts. On the other hand, the upper layers including the applications can indicate the credit as the safe band time that allows for the overlayed path segment to do the retransmission. At the same time, the persistent congestion caused packet loss should be exposed to the upper transport layer, so that the sender's congestion control can work properly. The timing of activation of the enhancement scheme, parameters such as the threshold setting of retransmission are worthy of further determination.

- Measurement based path selection for better performance, backup or load balancing. Overlay nodes have to be continuously monitored in order to find one or more appropriate overlayed paths. Such measurement can be in-band or out of band of data packets. When more than one overlayed segment with the same ingress and egress are used,

it has to be determined how the traffic are split and merged.

5. Security Considerations

TBD

6. IANA Considerations

No IANA action is required.

7. References

7.1 Normative References

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

7.2 Informative References

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, July 2005.

[RFC5415] Calhoun, P., Ed., Montemurro, M., Ed., and D. Stanley, Ed., "Control And Provisioning of Wireless Access Points (CAPWAP) Protocol Specification", RFC 5415, March 2009.

[BurstyAna] Chung S., Agrawal D., Kim M., Hong J., and Park K. "Analysis of bursty packet loss characteristics on underutilized links using SNMP", IEEE/IFIP E2EMON, 2004.

[CRONets] Cai, C. X., Le, F., Sun, X., Xie, G. G., Jamjoom, H., and Campbell, R. H. CRONets: Cloud-Routed Overlay Networks. In 36th International Conference on Distributed Computing Systems (ICDCS) (2016), IEEE, pp. 67--77.

[RFC6830] Farinacci, D., Fuller, V., Meyer, D., and D. Lewis, "The Locator/ID Separation Protocol (LISP)", RFC 6830, January 2013.

[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.

Authors' Addresses

Yizhou Li
Huawei Technologies
101 Software Avenue,
Nanjing 210012
China

Phone: +86-25-56624584
EMail: liyizhou@huawei.com

Xingwang Zhou
Huawei Technologies
101 Software Avenue,
Nanjing 210012
China

EMail: zhouxingwang@huawei.com

Internet Area Working Group V. Olteanu
Internet-Draft D. Niculescu
Intended status: Experimental University Politehnica of Bucharest
Expires: April 26, 2019 October 23, 2018

SOCKS Protocol Version 6
draft-olteanu-intarea-socks-6-05

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 April 26, 2019.

Table of Contents

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 satelite.

The desire to reduce the number of RTTs manifests itself in the design of newer security protocols. TLS version 1.3 [RFC8446] 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:

- 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.
- 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.
- The protocol can be extended via options without breaking backward-compatibility.
- 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-05

- Limited the "slow" authentication negociations to one (and Authentication Replies to 2)
- Revamped the handling of the first bytes in the application data stream
  - False starts are now recommended. (Added the "False Start" section.)
  - Initial data is only available to clients willing to do "slow" authentication. Moved the "Initial data size" field from Requests to Authentication Method options.
  - Initial data size capped at 2^13. Initial data can no longer be dropped by the proxy.
  - The TFO option can hint at the desired SYN payload size.
- Request: clarified the meaning of the Address and Port fields.
- Better reverse TCP proxy support: optional listen backlog for TCP BIND
- TFO options can no longer be placed inside Operation Replies.
- IP TOS stack option
- Suggested a range for vendor-specific options.
- Revamped UDP functionality
  - Now using fixed UDP ports
  - DTLS support
- Stack options: renamed Proxy-Server leg to Proxy-Remote leg

draft-04

- Moved Token Expenditure Replies to the Authentication Reply.

- Shifted the Initial Data Size field in the Request, in order to make it easier to parse.

draft-03

- Shifted some fields in the Operation Reply to make it easier to parse.
- Added connection attempt timeout response code to Operation Replies.
- Proxies send an additional Authentication Reply after the authentication phase. (Useful for token window advertisements.)
- Renamed the section "Connection Requests" to "Requests"
- Clarified the fact that proxies don’t need to support any command in particular.
- Added the section "TCP Fast Open on the Client-Proxy Leg"
- Options:
  - Added constants for option kinds
  - Salt options removed, along with the relevant section from Security Considerations. (TLS 1.3 Makes AEAD mandatory.)
  - Limited Authentication Data options to one per method.
  - Relaxed proxy requirements with regard to handling multiple Authentication Data options. (When the client violates the above bullet point.)
  - Removed interdependence between Authentication Method and Authentication Data options.
  - Clients SHOULD omit advertising the "No authentication required" option. (Was MAY.)
  - Idempotence options:
    - Token Window Advertisements are now part of successful Authentication Replies (so that the proxy-server RTT has no impact on their timeliness).
    - Proxies can’t advetise token windows of size 0.

        + Tweaked token expenditure response codes.

        + Support no longer mandatory on the proxy side.

    * Revamped Socket options

        + Renamed Socket options to Stack options.

        + Banned contradictory socket options.

        + Added socket level for generic IP. Removed the "socket" socket level.

        + Stack options no longer use option codes from setsockopt().

        + Changed MPTCP Scheduler constants.

draft-02

- Made support for Idempotence options mandatory for proxies.
- Clarified what happens when proxies can not or will not issue tokens.
- Limited token windows to 2^31 - 1.
- Fixed definition of "less than" for tokens.
- NOOP commands now trigger Operation Replies.
- Renamed Authentication options to Authentication Data options.
- Authentication Data options are no longer mandatory.
- Authentication methods are now advertised via options.
- Shifted some Request fields.
- Option range for vendor-specific options.
- Socket options.
- Password authentication.
- Salt options.

draft-01

- Added this section.
- Support for idempotent commands.
- Removed version numbers from operation replies.
- Request port number for SOCKS over TLS. Deprecate encryption/ encapsulation within SOCKS.
- Added Version Mismatch Replies.
- Renamed the AUTH command to NOOP.
- 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

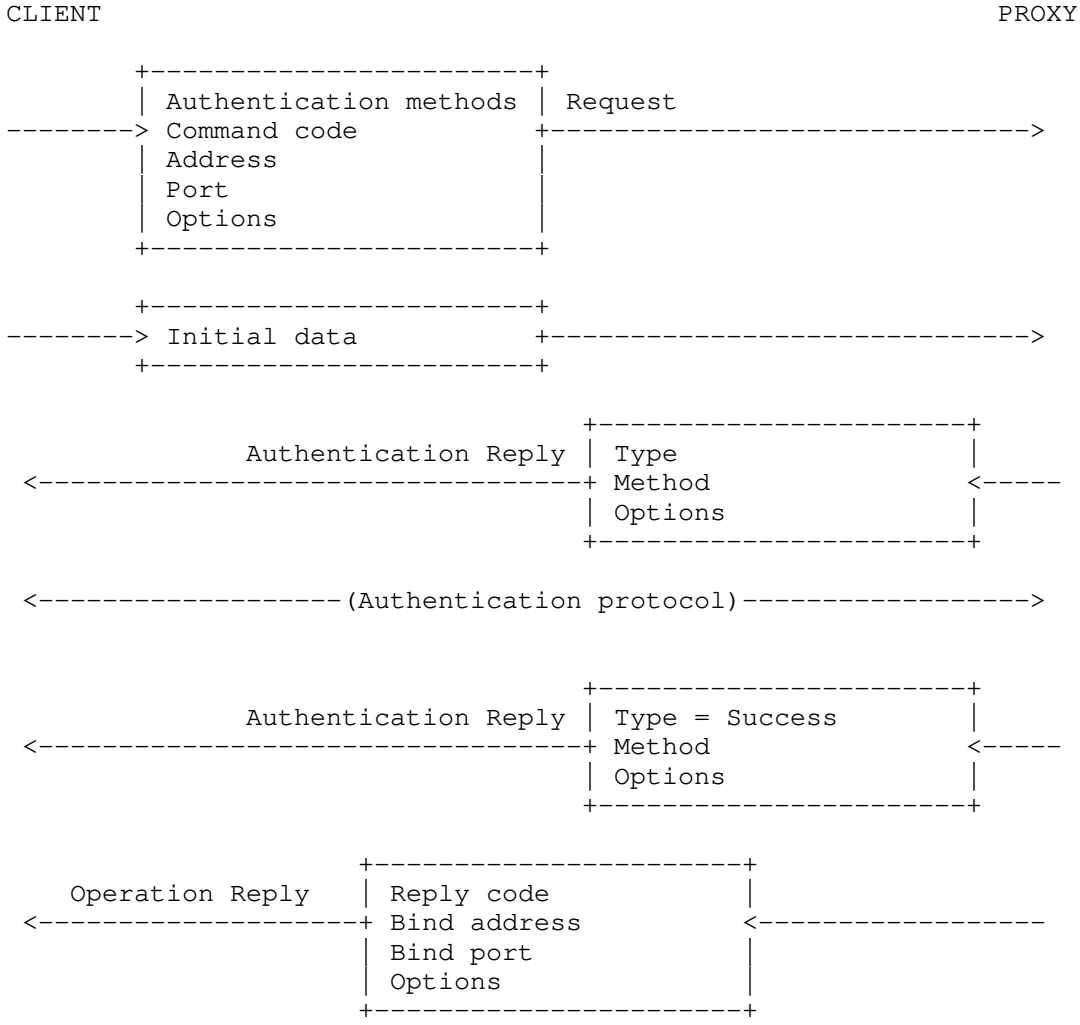


Figure 1: The SOCKS version 6 protocol message exchange

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 conection in one RTT (provided TFO functionality is available at the client, proxy, and server).

4. Requests

The client starts by sending a request to the proxy.

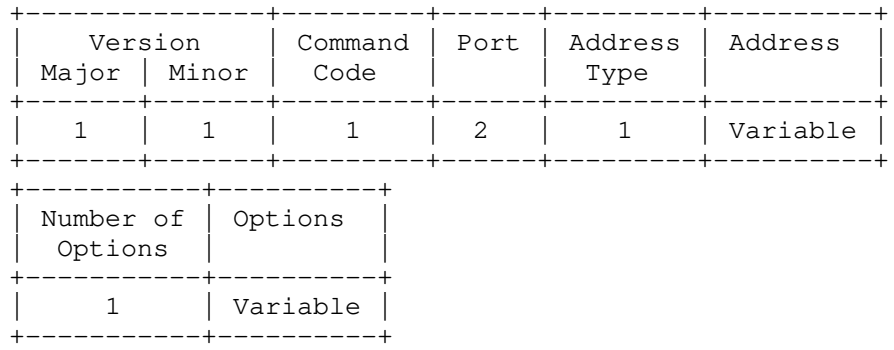


Figure 2: SOCKS 6 Request

- Version: The major byte MUST be set to 0x06, and the minor byte MUST be set to 0x00.
- 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.

- Address Type:
  * 0x01: IPv4
  * 0x03: Domain Name
  * 0x04: IPv6
- 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
- Port: the port in network byte order.
- Number of Options: the number of SOCKS options that appear in the Options field.
- Options: see Section 8.

The Address and Port fields have different meanings based on the Command Code: * NOOP: The fields have no meaning. The Address Type field MUST be either 0x01 (IPv4) or 0x04 (IPv6). The Address and Port fields MUST be 0. * CONNECT: The fields signify the address and port to which the client wishes to connect. * BIND, UDP ASSOCIATE: The fields indicate the desired bind address and port. If the client does not require a certain address, it can set the Address Type field to 0x01 (IPv4) or 0x04 (IPv6), and the Address field to 0. Likewise, if the client does not require a certain port, it can set the Port field to 0.

Clients can advertise their supported authentication methods by including an Authentication Method option (see Section 8.2).

5. Version Mismatch Replies

Upon receipt of a request starting with a version number other than 6.0, the proxy sends the following response:

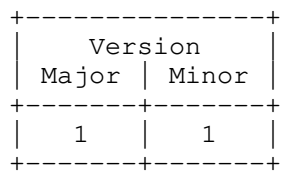


Figure 3: SOCKS 6 Version Mismatch Reply

- 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:

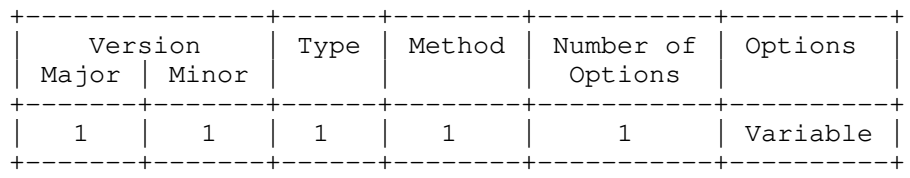


Figure 4: SOCKS 6 Authentication Reply

- Version: The major byte MUST be set to 0x06, and the minor byte MUST be set to 0x00.

- Type:

  * 0x00: authentication successful.

  * 0x01: further authentication needed.

- Method: The chosen authentication method.

- Number of Options: the number of SOCKS options that appear in the Options field.

- 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.

When the negotiation is complete (either successfully or unsuccessfully), the proxy sends a second Authentication Reply. The second Authentication Reply MUST either signal success or that there are no more acceptable authentication methods.

7. Operation Replies

After the authentication negotiations are complete, the proxy sends an Operation Reply:

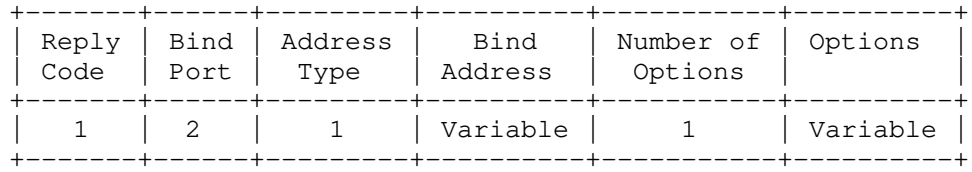


Figure 5: SOCKS 6 Operation Reply

- Reply Code:
  - 0x00: Succes
  - 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

    * 0x09: Connection attempt timed out

  o Bind Port: the proxy bound port in network byte order.

  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 Number of Options: the number of SOCKS options that appear in the Options field.

  o Options: see Section 8.

Proxy implementations MAY support any subset of the client commands listed in Section 4.

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.

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

Proxies offering UDP functionality must be configured with a UDP port used for relaying UDP datagrams to and from the client, and/or a port used for relaying datagrams over DTLS.

Following a successful Operation Reply, the proxy sends a UDP Association Initialization message:

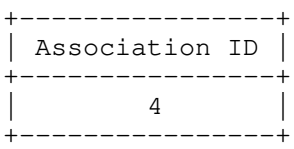


Figure 6: UDP Association Initialization

- Association ID: the identifier of the UDP association

Proxy implementations SHOULD generate Association IDs randomly or pseudo-randomly.

Clients may start sending UDP datagrams to the proxy either in plaintext, or over an established DTLS session, using the proxy's configured UDP ports. A client's datagrams are prefixed by a SOCKS Datagram Header, indicating the remote host's address and port:

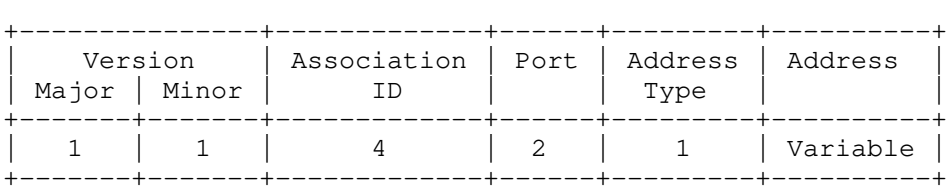


Figure 7: SOCKS 6 Datagram Header

- Version: The major byte MUST be set to 0x06, and the minor byte MUST be set to 0x00.

- Association ID: the identifier of the UDP association
- Address Type:
  - 0x01: IPv4
  - 0x03: Domain Name
  - 0x04: IPv6
- 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
- Port: the port in network byte order.

Following the receipt of the first datagram from the client, the proxy makes a one-way mapping between the Association ID and:

- the 5-tuple of the UDP conversation, if the datagram was received over plain UDP, or
- the DTLS connection, if the datagram was received over DTLS. The DTLS connection is identified either by its 5-tuple, or some other mechanism, like [I-D.ietf-tls-dtls-connection-id].

Further datagrams carrying the same Association ID, but not matching the established mapping, are silently dropped.

The proxy then sends an UDP Association Confirmation message over the TCP connection with the client:

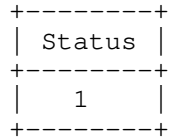


Figure 8: UDP Association Confirmation

- Status: MUST be 0x00

Following the confirmation message, UDP packets bound for the proxy's bind address and port are relayed to the client, also prefixed by a Datagram Header.

The UDP association remains active for as long as the TCP connection between the client and the proxy is kept open.

7.3.1. Proxying UDP servers

Under some circumstances (e.g. when hosting a server), the SOCKS client expects the remote host to send UDP datagrams first. As such, the SOCKS client must trigger a UDP Association Confirmation without having the proxy relay any datagrams on its behalf.

To that end, it sends an empty datagram prefixed by a Datagram Header with an IP address and port consisting of zeroes. The client SHOULD resend the empty datagram if an UDP Association Confirmation is not received after a timeout.

8. SOCKS Options

SOCKS options have the following format:

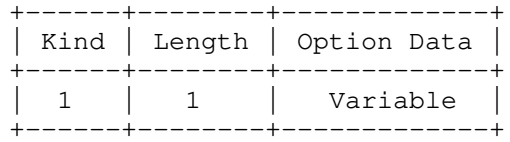


Figure 9: SOCKS 6 Option

- Kind: MUST be allocated by IANA. (See Section 13.)
- Length: The length of the option.
- Option Data: The contents are specific to each option kind.

Unless otherwise noted, client and proxy implementations MAY omit supporting any of the options described in this document. Upon encountering an unsupported option, a SOCKS endpoint MUST silently ignore it.

8.1. Stack options

Stack options can be used by clients to alter the behavior of the protocols on top of which SOCKS is running, as well the protcols used by the proxy to communicate with the remote host (i.e. IP, TCP,

UDP). A Stack option can affect either the proxy's protocol on the client-proxy leg or on the proxy-remote leg. Clients can only place Stack options inside SOCKS Requests.

Proxies MAY include Stack options in their Operation Replies to signal their behavior. Said options MAY be unsolicited, i. e. the proxy MAY send them to signal behaviour that was not explicitly requested by the client.

In case of UDP ASSOCIATE, the stack options refer to the UDP traffic relayed by the proxy.

Stack options that are part of the same message MUST NOT contradict one another.

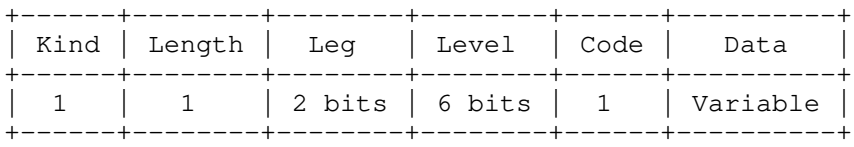


Figure 10: Stack Option

- Kind: 0x01 (Stack option)
- Length: The length of the option.
- Leg:
  - 0x1: Client-Proxy Leg
  - 0x2: Proxy-Remote Leg
  - 0x3: Both Legs
- Level:
  - 0x01: IP
  - 0x02: IPv4
  - 0x03: IPv6
  - 0x04: TCP
  - 0x05: UDP

- Code: Option code
- Data: Option-specific data

8.1.1. IP TOS options

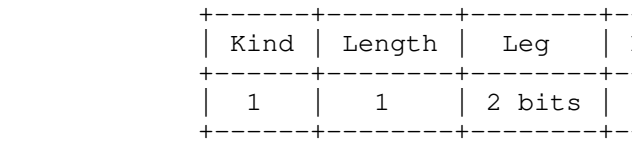


Figure 11: IP TOS Option

- Kind: 0x01 (Stack option)
- Length: 4
- Leg: Either 0x01, 0x02, or 0x03 (Client-Proxy, Proxy-Remote or Both legs)
- Level: 0x04 (TCP).
- Code: 0x01

The client can use IP TOS options to request that the proxy use a certain value for the IP TOS field. Likewise, the proxy can use IP TOS options to advertise the TOS values being used.

8.1.2. TFO options

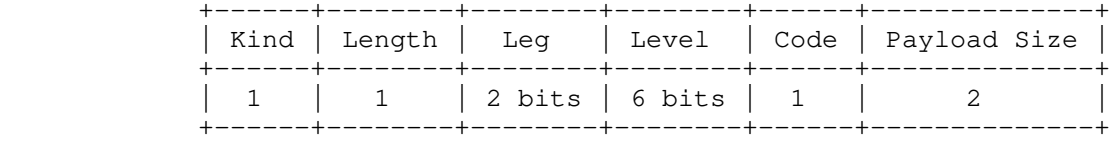


Figure 12: TFO Option

- Kind: 0x01 (Stack option)
- Length: 4
- Leg: 0x2 (Proxy-Remote leg).
- Level: 0x04 (TCP).

- Code: 0x01

- Payload Size: The desired payload size of the TFO SYN. MUST be 0 in case of a BIND command.

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 client can indicate the desired payload size of the SYN. The proxy MAY use a different payload size than the one indicated.

8.1.3. 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.

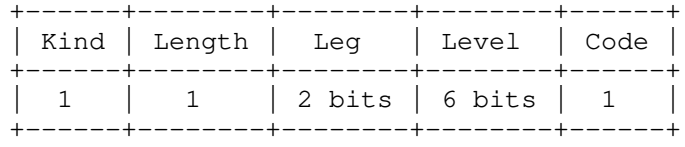


Figure 13: Multipath TCP Option

- Kind: 0x01 (Stack option)

- Length: 4

- Leg: 0x2 (Proxy-Remote leg)

- Level: 0x04 (TCP).

- Code: 0x02

8.1.4. 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.

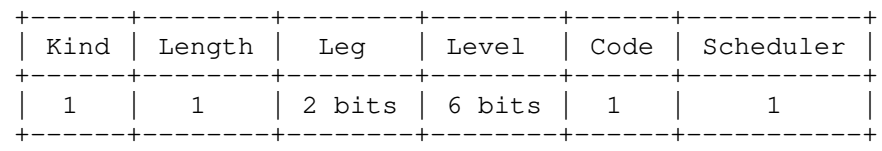


Figure 14: MPTCP Scheduler Option

- Kind: 0x01 (Stack option)
- Length: 5
- Leg: Either 0x01, 0x02, or 0x03 (Client-Proxy, Proxy-Remote or Both legs).
- Level: 0x04 (TCP)
- Code: 0x03
- Scheduler:
  - 0x01: Default
  - 0x02: Round-Robin
  - 0x03: Redundant

8.1.5. Listen Backlog options

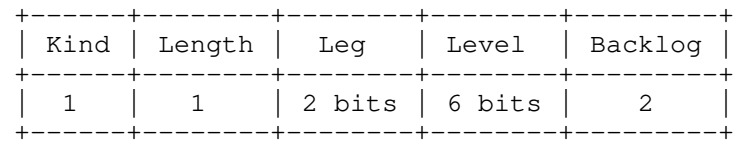


Figure 15: Listen Backlog Option

- Kind: 0x01 (Stack option)
- Length: 5
- Leg: 0x02 (Proxy-Remote leg)
- Level: 0x04 (TCP)
- Code: 0x04

- Backlog: The length of the listen backlog. MUST be greater than 1.

The default behavior of the BIND does not allow a client to simultaneously handle multiple connections to the same bind address. An authenticated client can alter BIND's behavior by adding a TCP Listen Backlog Option to a BIND Request.

In response, the proxy sends a TCP Listen Backlog Option as part of the Operation Reply, with the Backlog field signalling the actual backlog used. The proxy SHOULD NOT use a backlog longer than requested.

Following the successful negotiation of a backlog, the proxy listens for incoming connections for as long as the initial connection stays open. The initial connection is not used to relay data between the client and a remote host.

To accept connections, the client issues further BIND Requests using the bind address and port supplied by the proxy in the initial Operation Reply.

8.2. Authentication Method options

Authentication Method options are placed in SOCKS Requests to advertise supported authentication methods. In case of a CONNECT Request, they are also used to specify the amount of initial data supplied before any method-specific authentication negotiations take place.

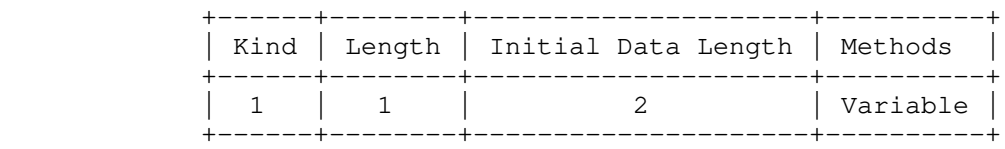


Figure 16: Authentication Method Option

- Kind: 0x02 (Authentication Method option)

- Length: The length of the option.

- Initial Data Size: A two-byte number in network byte order. In case of CONNECT, this is the number of bytes of initial data that are supplied by the client immediately following the Request. This number MUST NOT be larger than 2^13.

- Methods: One byte per advertised method. Method numbers are assigned by IANA.

Clients MUST support the "No authentication required" method. Clients SHOULD 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:

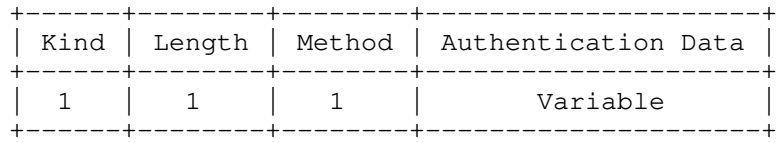


Figure 17: Authentication Data Option

- Kind: 0x03 (Authentication Data option)

- Length: The length of the option.

- Method: The number of the authentication method. These numbers are assigned by IANA.

- Authentication Data: The contents are specific to each method.

Clients SHOULD only place one Authentication Data option per authentication method. Server implementations MAY silently ignore all Authentication Data options for the same method aside from an arbitrarily chosen one.

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:

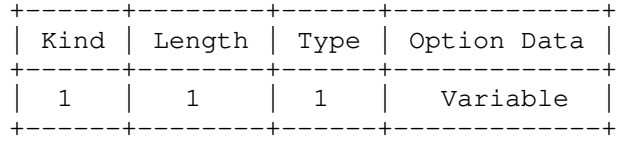


Figure 18: Idempotence Option

- Kind: 0x04 (Idempotence option)
- Length: The length of the option.
- Type:
  - 0x00: Token Request
  - 0x01: Token Window Advertisement
  - 0x02: Token Expenditure
  - 0x03: Token Expenditure Reply
- Option Data: The contents are specific to each type.

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:

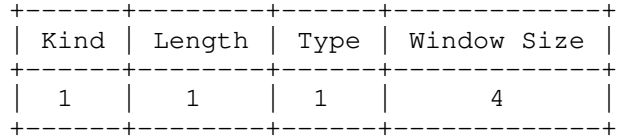


Figure 19: Token Request

- Kind: MUST be allocated by IANA. (See Section 13.)

- Length: 7
- Type: 0x00 (Token Request)
- Window Size: The requested window size.

If a token window is issued, the proxy then includes a Token Window Advertisement option in the corresponding successful Authentication Reply:

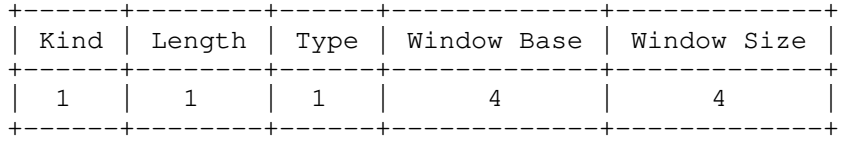


Figure 20: Token Window Advertisement

- Kind: 0x04 (Idempotence option)
- Length: 11
- Type: 0x01 (Token Grant)
- Window Base: The first token in the window.
- 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. Window sizes MUST NOT be 0.

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:

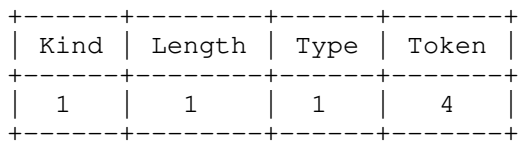
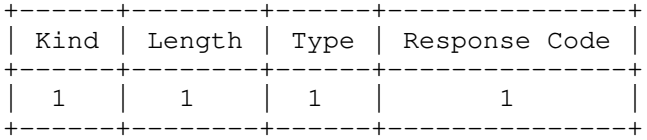


Figure 21: Token Expenditure

- Kind: 0x04 (Idempotence option)
- Length: 7
- Type: 0x02 (Token Expenditure)
- Token: The token being spent.

Clients SHOULD prioritize spending the smaller tokens.

The proxy responds by sending a Token Expenditure Reply option as part of the successful Authentication Reply:



Kind	Length	Type	Response Code
1	1	1	1

Figure 22: Token Expenditure Response

- Kind: 0x04 (Idempotence option)
- Length: 4
- Type: 0x03 (Token Expenditure Response)
- Response Code:
  - 0x01: Success: The token was spent successfully.
  - 0x02: No Window: The proxy does not have a token window associated with the client.
  - 0x03: Out of Window: The token is not within the window.
  - 0x04: 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:

- the token is out of window, or

- 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 Authentication Reply that indicates success.

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.

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.

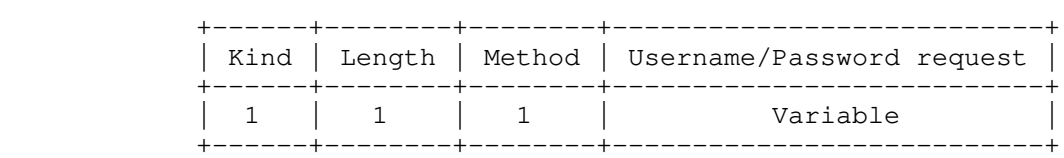


Figure 23: Password authentication via a SOCKS Option

- Kind: MUST be allocated by IANA. (See Section 13.)
- Length: The length of the option.
- Method: 0x02 (Username/Password).
- Username/Password request: The Username/Password request, as described in [RFC1929].

10. TCP Fast Open on the Client-Proxy Leg

TFO breaks TCP semantics, causing replays of the data in the SYN's payload under certain rare circumstances [RFC7413]. A replayed SOCKS

Request could itself result in a replayed connection on behalf of the client.

As such, client implementations SHOULD NOT use TFO on the client-proxy leg unless:

- The protocol running on top of SOCKS tolerates the risks of TFO, or
- The SYN's payload does not contain any application data (so that no data is replayed to the server, even though duplicate connections are still possible), or
- The client uses Idempotence Options, making replays impossible, or
- SOCKS is running on top of TLS and Early Data is not used.

11. False Starts

In case of CONNECT Requests, the client MAY start sending application data as soon as possible, as long as doing so does not incur the risk of breaking the SOCKS protocol.

Clients must work around the authentication phase by doing any of the following:

- If the Request does not contain an Authentication Method option, the authentication phase is guaranteed not to happen. In this case, application data MAY be sent immediately after the Request.
- Application data MAY be sent immediately after receiving an Authentication Reply indicating success.
- When performing a method-specific authentication sequence, application data MAY be sent immediately after the last client message.

12. Security Considerations

12.1. Large requests

Given the format of the request message, a malicious client could craft a request that is in excess of 80 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:

- the request is not fully received after a certain timeout, or
- the number of options exceeds an imposed hard cap, or
- the total size of the options exceeds an imposed 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.

12.2. 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.

13. IANA Considerations

This document requests that IANA allocate 1-byte option kinds for SOCKS 6 options. Further, this document requests the following option kinds:

- Stack options: 0x01
- Authentication Method options: 0x02
- Authentication Data options: 0x03
- Idempotence options: 0x04
- A range for vendor-specific options: 0xC0-0xFF

This document also requests that IANA allocate a TCP and UDP port for SOCKS over TLS and DTLS, respectively.

14. Acknowledgements

The protocol described in this draft builds upon and is a direct continuation of SOCKS 5 [RFC1928].

15. References

15.1. 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>.

15.2. Informative References

[I-D.ietf-tls-dtls-connection-id] Rescorla, E., Tschofenig, H., Fossati, T., and T. Gondrom, "Connection Identifiers for DTLS 1.2", draft-ietf-tls-dtls-connection-id-02 (work in progress), October 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>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

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

INTAREA
Internet Draft
Intended status: Standards Track
Expires: April 19,2019

J. Zhu
Intel
S. Seo
Korea Telecom
S. Kanugovi
Nokia
S. Peng
Huawei

October 19, 2018

User-Plane Protocols for Multiple Access Management Service
draft-zhu-intarea-mams-user-protocol-06

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/1id-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 April 19, 2009.

Abstract

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

Table of Contents

1. Introduction

Multi Access Management Service (MAMS) [MAMS] is a programmable framework to select and configure network paths, as well as adapt to dynamic network conditions, when multiple network connections can serve a client device. It is based on principles of user plane interworking that enables the solution to be deployed as an overlay without impacting the underlying networks.

This document presents the u-plane protocols for enabling the MAMS framework. It co-exists and complements the existing protocols by providing a way to negotiate and configure the protocols based on client and network capabilities. Further it allows exchange of network state information and leveraging network intelligence to optimize the performance of such protocols. An important goal for MAMS is to ensure that there is minimal or no dependency on the actual access technology of the participating links. This allows the scheme to be scalable for addition of newer access technologies and for independent evolution of the existing access technologies.

2. Terminologies

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

Delivery Connection: refers to the network path from the N-MADP to the C-MADP.

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

3. Conventions used in this document

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

The terminologies "Network Connection Manager" (NCM), "Client Connection Manager" (CCM), "Network Multi Access Data Proxy" (N-MADP), and "Client Multi Access Data Proxy" (C-MADP) in this document are to be interpreted as described in [MAMS].

4 MAMS User-Plane Protocols

Figure 1 shows the MAMS u-plane protocol stack as specified in [MAMS_CP].

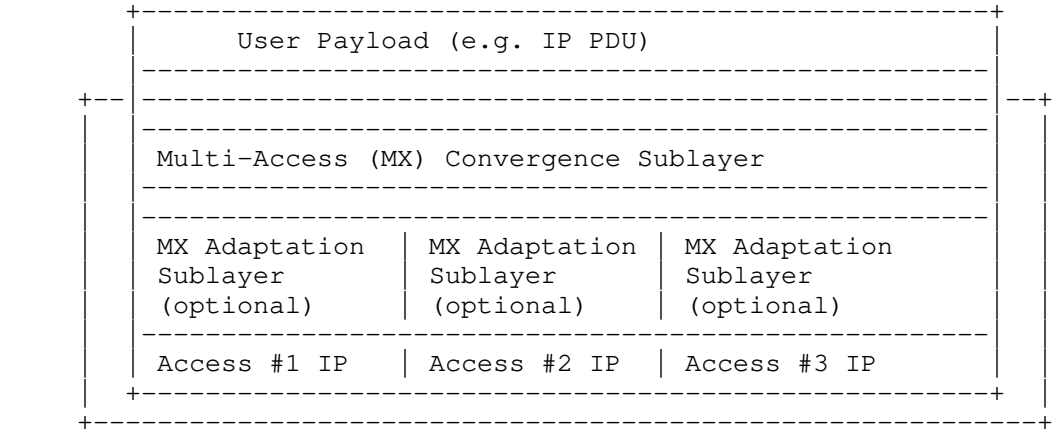


Figure 1: MAMS U-plane Protocol Stack

It consists of the following two Sublayers:

- Multi-Access (MX) Convergence Sublayer: This layer performs multi-access specific tasks, e.g., access (path) selection, multi-link (path) aggregation, splitting/reordering, lossless switching, fragmentation, concatenation, keep-alive, and probing etc.
- Multi-Access (MX) Adaptation Sublayer: This layer performs functions to handle tunneling, network layer security, and NAT.

The MX convergence sublayer operates on top of the MX adaptation sublayer in the protocol stack. From the Transmitter perspective, a User Payload (e.g. IP PDU) is processed by the convergence sublayer first, and then by the adaptation sublayer before being transported over a delivery access connection; from the Receiver perspective, an IP packet received over a delivery connection is processed by the MX adaptation sublayer first, and then by the MX convergence sublayer.

4.1 MX Adaptation Sublayer

The MX adaptation sublayer supports the following mechanisms and protocols while transmitting user plane packets on the network path:

- o UDP Tunneling: The user plane packets of the anchor connection can be encapsulated in a UDP tunnel of a delivery connection between the N-MADP and C-MADP.
- o IPsec Tunneling: The user plane packets of the anchor connection are sent through an IPsec tunnel of a delivery connection.
- o Client Net Address Translation (NAT): The Client IP address of user plane packet of the anchor connection is changed, and sent over a delivery connection.
- o Pass Through: The user plane packets are passing through without any change over the anchor connection.

The MX adaptation sublayer also supports the following mechanisms and protocols to ensure security of user plane packets over the network path.

- o IPsec Tunneling: An IPsec [RFC7296] tunnel is established between the N-MADP and C-MADP on the network path that is considered untrusted.
- o DTLS: If UDP tunneling is used on the network path that is considered "untrusted", DTLS (Datagram Transport Layer Security) [RFC6347] can be used.

The Client NAT method is the most efficient due to no tunneling overhead. It SHOULD be used if a delivery connection is "trusted" and without NAT function on the path.

The UDP or IPsec Tunnelling method SHOULD be used if a delivery connection has a NAT function placed on the path.

4.2 Trailer-based MX Convergence Sublayer

4.2.1 Trailer-based MX PDU Format

Trailer-based MX convergence integrates multiple connections into a single e2e IP connection. It operates between Layer 2 (L2) and Layer 3 (network/IP).

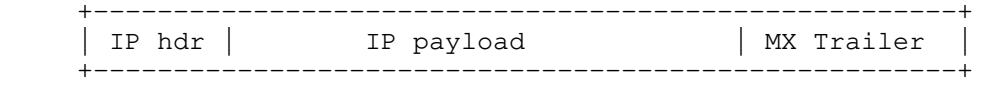


Figure 2: Trailer-based Multi-Access (MX) Data PDU Format

Figure 2 shows the trailer-based Multi-Access (MX) PDU (Protocol Data Unit) format. A MX PDU MAY carry multiple IP PDUs in the payload if concatenation is supported, and MAY carry a fragment of the IP PDU if fragmentation is supported.

The MX trailer may consist of the following fields:

- MX flags (e.g. 1 Byte): Bit 0 is the most significant bit, bit 7 is the least significant bit. Bit 6 and 7 are reserved for future.
  - Next Header Present (bit 0): If the Next Header Present bit is set to 1, then the Next Header field is present and contains valid information.
  - Connection ID Present (bit 1): If the Connection ID Present bit is set to 1, then the Connection ID field is present and contains valid information.
  - Traffic Class Present (bit 2): If the Traffic Class Present bit is set to 1, then the Traffic Class field is present and contains valid information.
  - Sequence Number Present (bit 3): If the Sequence Number Present bit is set to 1, then the Sequence Number field is present and contains valid information.
  - Packet Length Present (bit 4): If the Packet Length Present bit is set to 1, then the First SDU (Service Data Unit) Length field is present and contains valid information.
  - Fragmentation Control Present (bit 5): If the Fragmentation Control Present bit is set to 1, then the Fragmentation Control field is present and contains valid information.
  - Traffic Splitting Flag (bit 6): The bit will be flipped (0 or 1) when the traffic splitting configuration changes
  - Bit 7: reserved
- Next Header (e.g. 1 Byte): the IP protocol type of the (first) IP packet in a MX PDU
- Connection ID (e.g.1 Byte): an unsigned integer to identify the anchor connection of the IP packets in a MX PDU
- Traffic Class (TC) ID (e.g. 1 Byte): an unsigned integer to identify the traffic class of the IP packets in a MX PDU, for example Data Radio Bearer (DRB) ID [LWIPEP] for a cellular (e.g. LTE) connection
- Sequence Number (e.g. 2 Bytes): an auto-incremented integer to indicate order of transmission of the MX SDU (e.g. IP packet), needed for lossless switching or multi-link (path) aggregation or fragmentation. Sequence Number SHALL be generated on a per Connection and per Traffic Class (TC) basis.
- First SDU Length (e.g. 2 Bytes): the length of the first IP packet, only included if a MX PDU contains multiple IP packets, i.e. concatenation.
- Fragmentation Control (FC) (e.g. 1 Byte): to provide necessary information for re-assembly, only needed if a MX PDU carries fragments, i.e. fragmentation.

Figure 3 shows the MX trailer format with all the fields present. The MX flags are always encoded in the last octet of the MX Trailer at the end of a MX PDU. Hence, the Receiver SHOULD first decode the MX flags

field to determine the length of the MX trailer, and then decode each MX field accordingly.

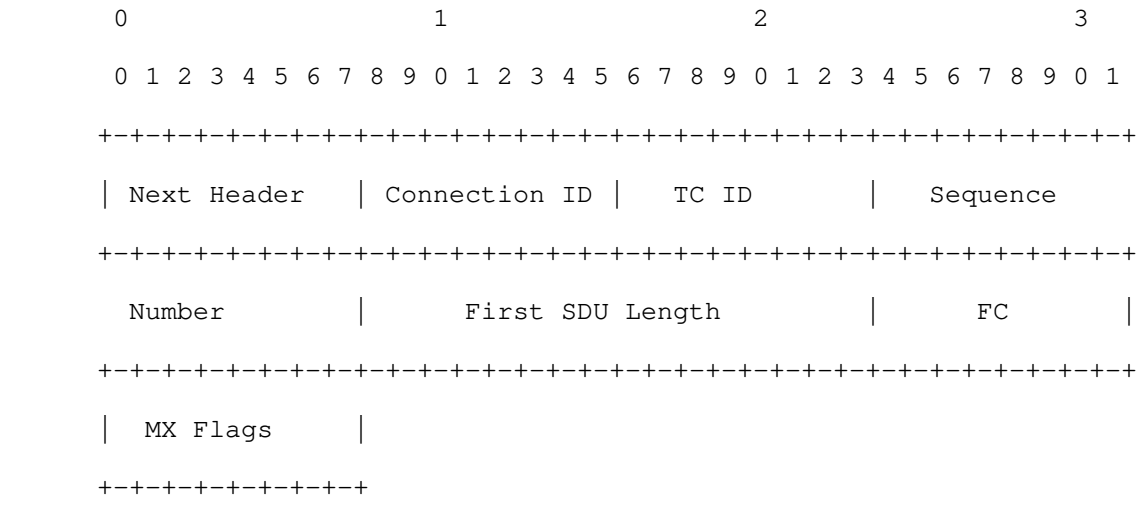


Figure 3: MX Trailer Format

Moreover, the following field of the IP header of the MX PDU SHOULD be changed:

o Protocol Type: "114" to indicate that the presence of MX trailer (i.e. the trailer based MAMS u-plane protocol is a "0-hop" protocol, not subject to IP routing)

If the MX PDU is transported with the MX adaptation method of IPSec tunnelling, Client NAT, or Pass Through, the following fields of the IP header of the MX PDU SHOULD also be changed:

o IP length: add the length of "MX Trailer" to the length of the original IP packet
o IP checksum: recalculate after changing "Protocol Type" and "IP Length"

If the MX adaptation method is UDP tunnelling and "MX header optimization" in the "MX_UP_Setup_Configuration_Request" message [MAMS] is true, the "IP length" and "IP checksum" header fields of the MX PDU SHOULD remain unchanged.

The MX u-plane protocol can support multiple Anchor connections simultaneously, each of which is uniquely identified by Connection ID. It can also support multiple traffic classes per connection, each of which is identified by Traffic Class ID.

Moreover, the MX trailer format MAY be negotiated dynamically between NCM and CCM. For example, NCM can send a control message to indicate which of the above fields SHOULD be included for individual delivery connection, on downlink and uplink, respectively.

4.2.2 MX Fragmentation

The Trailer-based MX Convergence Layer SHOULD support MX fragmentation if a delivery connection has a smaller maximum transmission unit (MTU) than the original IP packet (MX SDU), and IP fragmentation is not supported or enabled on the connection. The MX fragmentation procedure is similar to IP fragmentation [RFC791] in principle, but with the following two differences for less overhead:

- The fragment offset field is expressed in number of fragments not 8-bytes blocks
- The maximum number of fragments per MX SDU is 2^7 (=128)

The Fragmentation Control (FC) field in the MX Trailer contains the following bits:

- Bit #7: a More Fragment (MF) flag to indicate if the fragment is the last one (0) or not (1)
- Bit #0~#6: Fragment Offset (in units of fragments) to specify the offset of a particular fragment relative to the beginning of the MX SDU

A MX PDU carries a whole MX SDU without fragmentation if the FC field is set to all "0"s or the FC field is not present in the trailer. Otherwise, the MX PDU contains a fragment of the MX SDU.

The Sequence Number (SN) field in the trailer is used to distinguish the fragments of one MX SDU from those of another. The Fragment Offset (FO) field tells the receiver the position of a fragment in the original MX SDU. The More Fragment (MF) flag indicates the last fragment.

To fragment a long MX SDU, the MADP transmitter creates two MX PDUs and copies the content of the IP header fields from the long MX PDU into the IP header of both MX PDUs. The length field in the IP header of MX PDU SHOULD be changed to the length of the MX PDU, and the protocol type SHOULD be changed to "114", indicating the presence of the MX trailer.

The data of the long MX SDU is divided into two portions based on the MTU size of the delivery connection. The first portion of the data is placed in the first MX PDU. The MF flag is set to "1", and the FO field is set to "0". The second portion of the data is placed in the second

MX PDU. The MF flag is set to "0", and the FO field is set to "1". This procedure can be generalized for an n-way split, rather than the two-way split described the above.

To assemble the fragments of a MX SDU, the MADP receiver combines MX PDUs that all have the same MX Sequence Number (in the trailer). The combination is done by placing the data portion of each fragment in the relative order indicated by the Fragment Offset in that fragment's MX trailer. The first fragment will have the Fragment Offset set to "0", and the last fragment will have the More-Fragments flag reset to "0".

4.2.3 MX Concatenation

The Trailer-based MX Convergence Layer MAY support MX concatenation if a delivery connection has a larger maximum transmission unit (MTU) than the original IP packet (MX SDU). Only the MX SDUs with the same client address, the same anchor connection and the same Traffic Class MAY be concatenated.

If the MX adaptation method is IPSec tunnelling, Client NAT, or Pass Through, The First SDU Length (FSL) field SHOULD be included in the MX Trailer to indicate the length of the first MX SDU.

If the MX adaptation method is UDP tunneling and "MX header optimization" in the "MX_UP_Setup_Configuration_Request" message [MAMS] is true, the FSL field SHOULD not be present, or the entire MX trailer MAY not be present. The MADP receiver compares the IP length field of the MX PDU and the actual length of the MX PDU to determine if the MX PDU contains multiple MX SDUs. If the MX PDU is larger than what the IP length field indicates, the MX PDU contains multiple MX SDUs; otherwise, the MX PDU contains only one MX SDU. To concatenate two or more MX SDUs, the MADP transmitter creates one MX PDU and copies the content of the IP header field from the first MX SDU into the IP header of the MX PDU. The data of the first MX SDU is placed in the first portion of the data of the MX PDU. The whole second MX SDU is then placed in the second portion of the data of the MX PDU (Figure 4). The procedure continues till the MX PDU size reaches the MTU of the delivery connection. If the FSL field is present in the MX Trailer, the IP length field of the MX PDU SHOULD be updated to include all concatenated SDUs and the trailer, and the IP checksum field SHOULD be recalculated.

To disaggregate a MX PDU, the MADP receiver first obtains the length of the first MX SDU from the FSL field in the trailer, and decodes the first MX SDU. If the FSL field or the MX Trailer is not present, the MADP receiver obtains the length of the first MX SDU directly from the IP length field of the MX PDU. The MADP receiver then obtains the length of the second MX SDU based on the length field in the second MX

SDU IP header, and decodes the second MX SDU. The procedure continues till no byte is left in the MX PDU.

If a MX PDU contains multiple SDUs, the SN field in the MX trailer is for the last MX SDU, and the SN of other SDU carried by the same PDU can be obtained according to its order in the PDU. For example, if the SN field is 6 and a MX PDU contains 3 SDUs (IP packets), then the SN is 4, 5, and 6 for the first, second, and the last IP packet in the PDU, respectively.

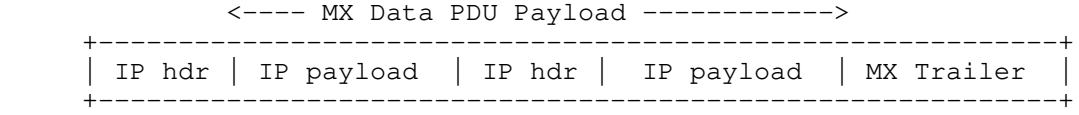


Figure 4: MX PDU Format with Concatenation

4.3 MPTCP-based MX Convergence Sublayer

Figure 5 shows the MAMS u-plane protocol stack based on MPTCP. Here, MPTCP is reused as the "MX Convergence Sublayer" protocol. Multiple access networks are combined into a single MPTCP connection. Hence, no new u-plane protocol or PDU format is needed in this case.

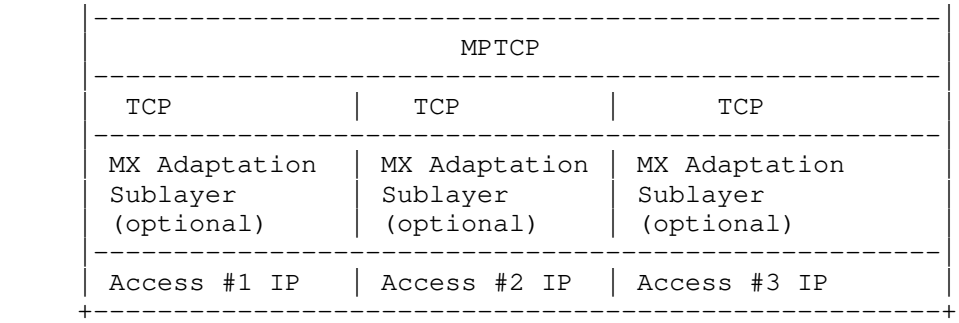


Figure 5: MAMS U-plane Protocol Stack with MPTCP as MX Convergence Layer

If NCM determines that N-MADP is to be instantiated with MPTCP as the MX Convergence Protocol, it exchanges the support of MPTCP capability in the discovery and capability exchange procedures [MAMS_CP]. MPTCP proxy protocols [MPProxy][MPPlain] SHOULD be used to manage traffic steering and aggregation over multiple delivery connections.

4.4 GRE as MX Convergence Sublayer

Figure 6 shows the MAMS u-plane protocol stack based on GRE (Generic Routing Encapsulation) [GRE2784]. Here, GRE is reused as the "MX Convergence sub-layer" protocol. Multiple access networks are combined into a single GRE connection. Hence, no new u-plane protocol or PDU format is needed in this case.

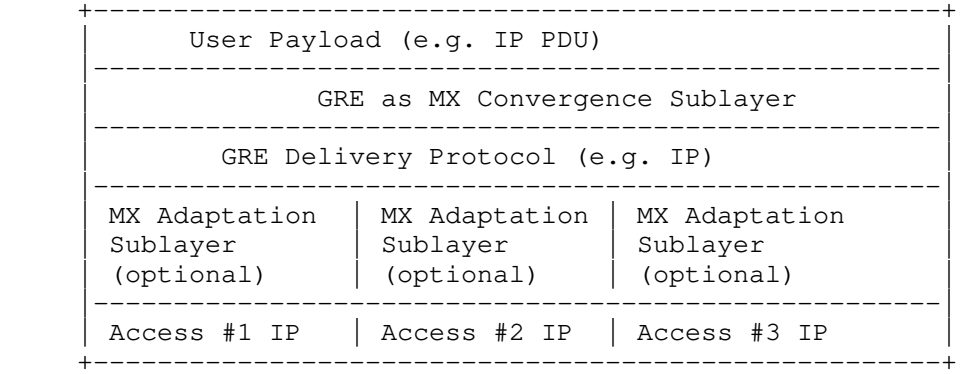


Figure 6: MAMS U-plane Protocol Stack with GRE as MX Convergence Layer

If NCM determines that N-MADP is to be instantiated with GRE as the MX Convergence Protocol, it exchanges the support of GRE capability in the discovery and capability exchange procedures [MAMS_CP].

4.4.1 Transmitter Procedures

Transmitter is the N-MADP or C-MADP instance, instantiated with GRE as the convergence protocol that transmits the GRE packets. The Transmitter receives the User Payload (e.g. IP PDU), encapsulates it with a GRE header and Delivery Protocol (e.g. IP) header to generate the GRE Convergence PDU.

When IP is used as the GRE delivery protocol, the IP header information (e.g. IP address) can be created using the IP header of the user payload or a virtual IP address. The "Protocol Type" field of the delivery header is set to 47 (or 0X2F, i.e. GRE)[IANA].

The GRE header fields are set as specified below,

- If the transmitter is a C-MADP instance, then sets the LSB 16 bits to the value of Connection ID for the Anchor Connection associated with the user payload or sets to 0xFFFF if no Anchor Connection ID needs to be specified.

- All other fields in the GRE header including the remaining bits in the key fields are set as per [GRE_2784][GRE_2890].

4.4.2 Receiver Procedures

Receiver is the N-MADP or C-MADP instance, instantiated with GRE as the convergence protocol that receives the GRE packets. The receiver processes the received packets per the GRE procedures [GRE_2784, GRE_2890] and retrieves the GRE header.

- If the Receiver is an N-MADP instance,
  - Unless the LSB 16 Bits of the Key field are 0xFFFF, they are interpreted as the Connection ID of Anchor Connection for the user payload. This is used to identify the network path over which the User Payload (GRE Payload) is to be transmitted.
- All other fields in the GRE header, including the remaining bits in the Key fields, are processed as per [GRE_2784][GRE_2890].

The GRE Convergence PDU is passed onto the MX Adaptation Layer (if present) before delivery over one of the network paths.

4.5 Co-existence of MX Adaptation and MX Convergence Sublayers

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

For example, one instance of the MX Convergence Layer can be MPTCP Proxy [MPProxy][MPPlain] and another instance can be Trailer-based. The MX Adaptation for each can be either UDP tunnel or IPsec. IPsec may be set up for network paths considered as untrusted by the operator, to protect the TCP subflow between client and MPTCP proxy traversing that network path.

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

5. MX Convergence Control Message

A UDP connection may be configured between C-MADP and N-MADP to exchange control messages for keep-alive or path quality estimation. The N-MADP end-point IP address and UDP port number of the UDP connection is used to identify MX control PDU. Figure 7 shows the MX control PDU format with the following fields:

o Type (1 Byte): the type of the MX control message
  + 0: Keep-Alive

+ 1: Probe REQ/ACK
+ 2: Packet Loss Report (PLR)
+ 3: First Sequence Number (FSN)
+ 4: Coded MX SDU (CMS)
+ 5: Traffic Splitting Update (TSU)
+ 6: Traffic Splitting Acknowledgement (TSA)
+ Others: reserved

o CID (1 Byte): the connection ID of the delivery connection for sending out the MX control message
o MX Control Message (variable): the payload of the MX control message

Figure 8 shows the MX convergence control protocol stack, and MX control PDU goes through the MX adaptation sublayer the same way as MX data PDU.

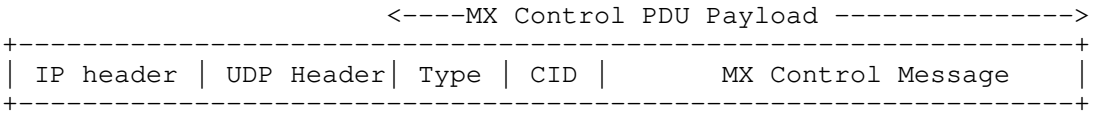


Figure 7: MX Control PDU Format

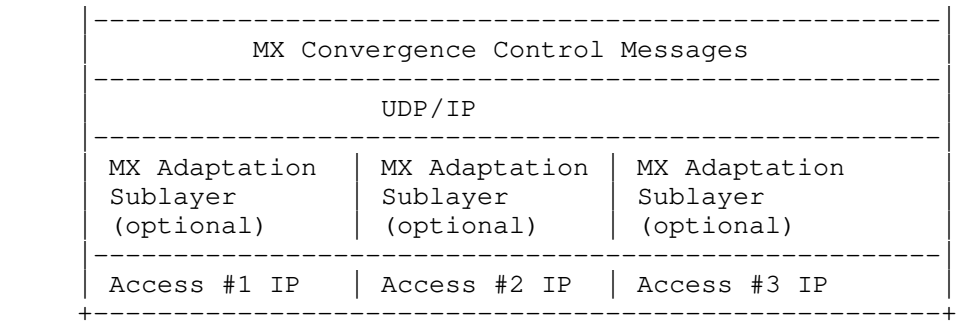


Figure 8: MX Convergence Control Protocol Stack

5.1 Keep-Alive Message

The "Type" field is set to "0" for Keep-Alive messages. C-MADP may send out Keep-Alive message periodically over one or multiple delivery connections, especially if UDP tunneling is used as the adaptation method for the delivery connection with a NAT function on the path.

A Keep-Alive message is 2 Bytes long, and consists of the following fields:

o Keep-Alive Sequence Number (2 Bytes): the sequence number of the keep-alive message

5.2 Probe REQ/ACK Message

The "Type" field is set to "1" for Probe REQ/ACK messages.

N-MADP may send out the Probe REQ message for path quality estimation. In response, C-MADP may send back the Probe ACK message.

A Probe REQ message consists of the following fields:

- Probing Sequence Number (2 Bytes): the sequence number of the Probe REQ message
- Probing Flag (1 Byte):
  - Bit #0: a Probe ACK flag to indicate if the Probe ACK message is expected (1) or not (0);
  - Bit #1: a Probe Type flag to indicate if the Probe REQ/ACK message is sent during the initialization phase (0) when the network path is not included for transmission of user data or the active phase (1) when the network path is included for transmission of user data;
  - Bit #2: a bit flag to indicate the presence of the Reverse Connection ID (R-CID) field.
  - Bit #3~7: reserved
- Reverse Connection ID (1 Byte): the connection ID of the delivery connection for sending out the Probe ACK message on the reverse path
- Padding (variable)

The "R-CID" field is only present if both Bit #0 and Bit #2 of the "Probing Flag" field are set to "1". Moreover, Bit #2 of the "Probing Flag" field SHOULD be set to "0" if the Bit #0 is "0", indicating the Probe ACK message is not expected.

If the "R-CID" field is not present but the Bit #0 of the "Probing Flag" field is set to "1", the Probe ACK message SHOULD be sent over the same delivery connection as the Probe REQ message.

The "Padding" field is used to control the length of Probe REQ message.

C-MADP SHOULD send out the Probe ACK message in response to a Probe REQ message with the Probe ACK flag set to "1".

A Probe ACK message is 2 Bytes long, and consists of the following fields:

- Probing Acknowledgement Number (2 Bytes): the sequence number of the corresponding Probe REQ message

5.3 Packet Loss Report (PLR) Message

The "Type" field is set to "2" for PLR messages.

C-MADP may send out the PLR messages to report lost MX SDU for example during handover. In response, C-MADP may retransmit the lost MX SDU accordingly.

A PLR message consists of the following fields:

- Connection ID (1 Byte): an unsigned integer to identify the anchor connection which the ACK message is for;
- Traffic Class ID (1 Byte): an unsigned integer to identify the traffic class of the anchor connection which the ACK message is for;
- ACK number (2 Bytes): the next (in-order) sequence number (SN) that the sender of the PLR message is expecting
- Number of Loss Bursts (1 Byte)
  For each loss burst, include the following
  + Sequence Number of the first lost MX SDU in a burst (2 Bytes)
  + Number of consecutive lost MX SDUs in the burst (1 Byte)

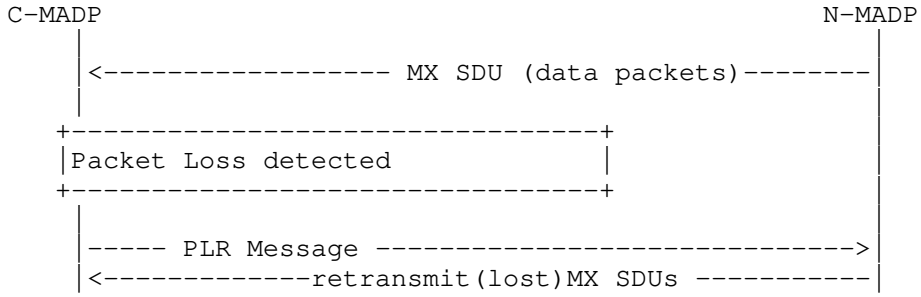


Figure 9: MAMS Retransmission Procedure

Figure 9 shows the MAMS retransmission procedure in an example where the lost packet is found and retransmitted.

5.4 First Sequence Number (FSN) Message

The "Type" field is set to "3" for FSN messages.

N-MADP may send out the FSN messages to indicate the oldest MX SDU in its buffer if a lost MX SDU is not found in the buffer after receiving

the PLR message from C-MADP. In response, C-MADP SHALL only report packet loss with SN not smaller than FSN.

A FSN message consists of the following fields:

- o Connection ID (1 Byte): an unsigned integer to identify the anchor connection which the FSN message is for;
- o Traffic Class ID (1 Byte): an unsigned integer to identify the traffic class of the anchor connection which the FSN message is for;
- o First Sequence Number (2 Bytes): the sequence number (SN) of the oldest MX SDU in the (retransmission) buffer of the sender of the FSN message.

Figure 10 shows the MAMS retransmission procedure in an example where the lost packet is not found.

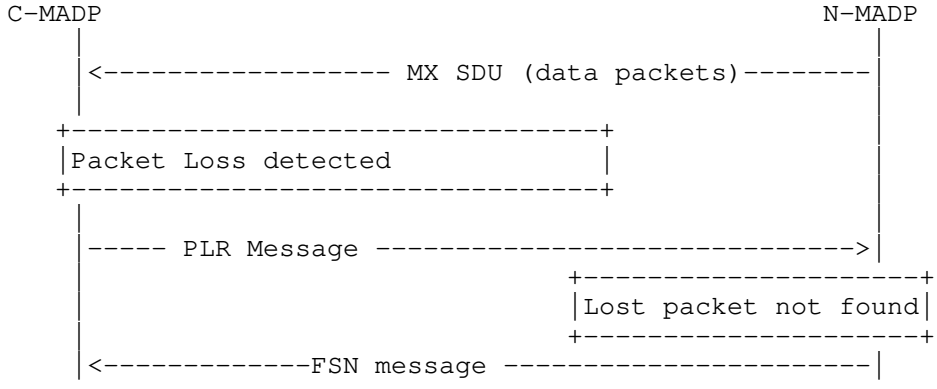


Figure 10: MAMS Retransmission Procedure with FSN

5.5 Coded MX SDU (CMS) Message

The "Type" field is set to "4" for CMS messages.

N-MADP (or C-MADP) may send out the CMS message to support downlink (or uplink) packet loss recovery through coding, e.g. [CRLNC], [CTCP], [RLNC]. A coded MX SDU is generated by applying a network coding algorithm to multiple consecutive (uncoded) MX SDUs, and it is used for fast recovery without retransmission if any of the MX SDUs is lost.

A Coded MX SDU message consists of the following fields:

- o Connection ID (1 Byte): an unsigned integer to identify the anchor connection of the coded MX SDU;
- o Traffic Class ID (1 Byte): an unsigned integer to identify the traffic class of the coded MX;
- o Sequence Number (2 Bytes): the sequence number of the first (uncoded) MX SDU used to generate the coded MX SDU.
- o Fragmentation Control (FC) (1 Byte): to provide necessary information for re-assembly, only needed if the coded MX SDU is too long to transport in a single MX control PDU.
- o N (1 Byte): the number of consecutive MX SDUs used to generate the coded MX SDU
- o K (1 Byte): the length (in terms of bits) of the coding coefficient field
- o Coding Coefficient (N x K / 8 Bytes)
  - + a(i): the coding coefficient of the i-th (uncoded) MX SDU
  - + padding
- o Coded MX SDU (variable): the coded MX SDU

If N = 2 and K = 0, the simple XOR method is used to generate the Coded MX SDU from two consecutive uncoded MX SDUs, and the a(i) fields are not included in the message.

If the coded MX SDU is too long, it can be fragmented, and transported by multiple MX control PDUs. The N, K, and a(i) fields are only included in the MX PDU carrying the first fragment of the coded MX SDU.

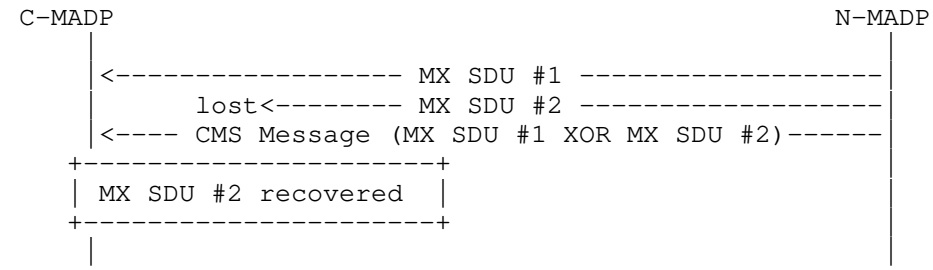


Figure 11: MAMS Packet Recovery Procedure with XOR Coding

5.6 Traffic Splitting Update (TSU) Message

The "Type" field is set to "5" for TSU messages.

N-MADP (or C-MADP) may send out a TSU message if downlink (or uplink) traffic splitting configuration has changed.

A TSU message consists of the following fields:

- o Connection ID (1 Byte): an unsigned integer to identify the anchor connection;
- o Traffic Class ID (1 Byte): an unsigned integer to identify the traffic class;
- o Sequence Number (2 Bytes): an unsigned integer to identify the TSU message.
- o Traffic Splitting Configuration Parameters (3 + (N -1) Bytes):
  - + StartSN (2 Bytes): the sequence number of the first MX SDU using the traffic splitting configuration provided by the TSU message
  - + L (1 Byte): the traffic splitting burst size
  - + K(i): the traffic splitting threshold of the i-th delivery connection, where connections are ordered according to their Connection ID.

Let's use f(x) to denote the traffic splitting function, which maps a MX SDU Sequence Number "x" to the i-th delivery connection.

f(x)=i, if K[i-1]< or = mod(x - StartSN, L) < K[i]

Wherein, 1 < or = i < N, K[0]=0, and K[N]=L.

N is the total number of connections for delivering a data flow, identified by (anchor) Connection ID and Traffic Class ID.

5.7 Traffic Splitting Acknowledgement (TSA) Message

The "Type" field is set to "6" for TSA messages.

C-MADP (or N-MADP) SHOULD send out the TSA message in response to the successful reception of a TSU message.

A TSU message consists of the following fields:

- o Connection ID (1 Byte): an unsigned integer to identify the anchor connection;
- o Traffic Class ID (1 Byte): an unsigned integer to identify the traffic class;
- o Acknowledgment Number (2 Bytes): the sequence number of the received TSU message.

6 Security Considerations

User data in MAMS framework rely on the security of the underlying network transport paths. When this cannot be assumed, NCM configures use of appropriate protocols for security, e.g. IPsec [RFC4301] [RFC3948], DTLS [RFC6347].

7 IANA Considerations

TBD

8 Contributing Authors

The editors gratefully acknowledge the following additional contributors in alphabetical order: Salil Agarwal/Nokia, Hema Pentakota/Nokia.

9 References

9.1 Normative References

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

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI10.17487/RFC4301, December 2005, <http://www.rfc-editor.org/info/rfc4301>.

9.2 Informative References

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

[RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <http://www.rfc-editor.org/info/rfc7296>.

[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>.

[MPProxy] X. Wei, C. Xiong, and E. Lopez, "MPTCP proxy mechanisms", https://tools.ietf.org/html/draft-wei-mptcp-proxy-mechanism-02

[MPPlain] M. Boucadair et al, "An MPTCP Option for Network-Assisted MPTCP", https://www.ietf.org/id/draft-boucadair-mptcp-plain-mode-09.txt

[MAMS] S. Kanugovi, S. Vasudevan, F. Baboescu, and J. Zhu, "Multiple Access Management Protocol", https://tools.ietf.org/html/draft-kanugovi-intarea-mams-protocol-03

[MAMS_CP] S. Kanugovi, et al., "Control Plane Protocols and Procedures for Multiple Access Management Services"

[GRE2784] D. Farinacci, et al., "Generic Routing Encapsulation (GRE)", RFC 2784 March 2000, <http://www.rfc-editor.org/info/rfc2784>.

[GRE2890] G. Dommety, "Key and Sequence Number Extensions to GRE", RFC 2890 September 2000, <http://www.rfc-editor.org/info/rfc2890>.

[IANA] https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml

[LWIPEP] 3GPP TS 36.361, "Evolved Universal Terrestrial Radio Access (E-UTRA); LTE-WLAN Radio Level Integration Using Ipsec Tunnel (LWIP) encapsulation; Protocol specification"

[RFC791] Internet Protocol, September 1981

[CRLNC] S Wunderlich, F Gabriel, S Pandi, et al. Caterpillar RLNC (CRLNC): A Practical Finite Sliding Window RLNC Approach, IEEE Access, 2017

[CTCP] M. Kim, et al. Network Coded TCP (CTCP), eprint arXiv:1212.2291, 2012

[RLNC] J. Heide, et al. Random Linear Network Coding (RLNC)-Based Symbol Representation, https://www.ietf.org/id/draft-heide-nwcrg-rlnc-00.txt

Authors' Addresses

Jing Zhu

Intel

Email: jing.z.zhu@intel.com

SungHoon Seo

Korea Telecom

Email: sh.seo@kt.com

Satish Kanugovi

Nokia

Email: satish.k@nokia.com

Shuping Peng

Huawei

Email: pengshuping@huawei.com