Stream:	Internet Engineering Task Force (IETF)	
RFC:	9868	
Updates:	768	
Category:	Standards Track	
Published:	September 2025	
ISSN:	2070-1721	
Authors:	J. Touch	C. Heard, Ed.
	Independent Consultant	*Unaffiliated*

RFC 9868
Transport Options for UDP

Abstract

Transport protocols are extended through the use of transport header options. This document updates RFC 768 (UDP) by indicating the location, syntax, and semantics for UDP transport layer options within the surplus area after the end of the UDP user data but before the end of the IP datagram.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9868.

Copyright Notice

Table of Contents

1. Introduction

Transport protocols use options as a way to extend their capabilities. TCP [RFC9293], the Stream Control Transmission Protocol (SCTP) [RFC9260], and the Datagram Congestion Control Protocol (DCCP) [RFC4340] include space for these options, but UDP [RFC0768] currently does not. This document updates RFC 768 with an extension to UDP that provides space for transport options

including their generic syntax and semantics for their use in UDP's stateless, unreliable message protocol. The details of the impact on RFC 768 are provided in Section 21. This extension does not apply to UDP-Lite, as discussed further in Section 17.

2. Conventions Used in This Document

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.

In this document, the characters ">>" preceding an indented line(s) indicate a statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the portions of this RFC covered by these key words.

3. Terminology

The following terminology is used in this document:

IP datagram [RFC0791] [RFC8200]: An IP packet, composed of the IP header (including any IPv4 options) and an IP payload area (including any IPv6 extension headers or other shim headers).

Must-support options: UDP options that all implementations are required to support. Their use in individual UDP packets is optional.

SAFE options: UDP options that are designed to be safe to ignore for a receiver that does not understand them. Such options do not alter the UDP user data or signal a change in what its contents represent.

Socket pair: A pair of sockets defining a UDP exchange, defined by a remote socket and a local socket, each composed of an IP address and UDP port number (most widely known from TCP [RFC0793]).

Surplus area: The area of an IP payload that follows a UDP packet; this area is used for UDP options in this document.

UDP packet: The more contemporary term used herein to refer to a user datagram [RFC0768].

UDP fragment: One or more components of a UDP packet and its UDP options that enable transmission over multiple IP payloads, larger than permitted by the maximum size of a single IP packet; note that each UDP fragment is itself transmitted as a UDP packet with its own options.

(UDP) User data: The user data field of a UDP packet [RFC0768].

UDP Length: The length field of a UDP header [RFC0768].

UNSAFE options: UDP options that are not designed to be safe for a receiver that does not understand them to ignore. Such options could alter the UDP user data or signal a change in what its contents represent, but there are restrictions on how they can be transmitted; these restrictions are noted in Sections 10 and 12.

User: The upper layer application, protocol, or service that produces and consumes content that UDP transfers.

User datagram: A UDP packet, composed of a UDP header and UDP payload; as discussed herein, that payload need not extend to the end of the IP datagram. In this document, the original intent that a UDP datagram corresponds to the user portion of a single IP datagram is redefined, where a UDP datagram can span more than one IP datagram through UDP fragmentation.

4. Background

Many protocols include a default, invariant header and an area for header options that varies from packet to packet. These options enable the protocol to be extended for use in particular environments or in ways unforeseen by the original designers. Examples include TCP's Maximum Segment Size (MSS), Window Scale, Timestamp, and Authentication Options [RFC9293] [RFC5925] [RFC7323].

Header options are used both in stateful (connection-oriented, e.g., TCP [RFC9293], SCTP [RFC9260], and DCCP [RFC4340]) and stateless (connectionless, e.g., IPv4 [RFC0791] and IPv6 [RFC8200]) protocols. In stateful protocols, they can help extend the way in which state is managed. In stateless protocols, their effect is often limited to individual packets, but they can have an aggregate effect on a sequence of packets as well.

UDP is one of the most popular protocols that lacks space for header options [RFC0768]. The UDP header was intended to be a minimal addition to IP, providing only port numbers and a checksum for error detection. This document extends UDP to provide a trailer area for such options, located after the UDP user data.

UDP options are possible because UDP includes its own length field, separate from that of the IP header. Other transport protocols infer transport payload length from the IP datagram length (TCP, DCCP, and SCTP). Internet historians have suggested a number of possible reasons why the design of UDP includes this field, e.g., to support multiple UDP packets within the same IP datagram or to indicate the length of the UDP user data as distinct from zero padding required for systems that require writes that are not byte-aligned. These suggestions are not consistent with earlier versions of UDP or with the concurrent design of multi-segment, multiplexing protocols; however, the real reason remains unknown. Regardless, this field presents an opportunity to differentiate the UDP user data from the implied transport payload length, which this document leverages to support a trailer options field.

There are other ways to include additional header fields or options in protocols that otherwise are not extensible. In particular, in- band encoding can be used to differentiate transport payload from additional fields, such as was proposed in [Hi15]. This approach can cause complications for interactions with legacy devices and is thus not considered further in this document.

IPv6 Teredo extensions (TEs) [RFC4380] [RFC6081] use a similar inconsistency between UDP and IPv6 packet lengths to support trailers, but in this case, the values differ between the UDP header and an IPv6 length contained as the payload of the UDP user data. This allows IPv6 trailers in the UDP user data but has no relation to the surplus area discussed in this document. As a consequence, TEs are compatible with UDP options.

5. UDP Option Intended Uses

UDP options can be used to provide a soft control plane to UDP. They enable capabilities available in other transport protocols, such as fragmentation and reassembly, that enable UDP frames larger than the IP MTU to traverse devices that rely on transport ports, e.g., Network Address Translations (NATs), without additional mechanisms or state. They add features that could, in the future, protect transport integrity and validate source identity (authentication), as well as those that could encrypt the user payload while still protecting the UDP transport header -- unlike Datagram Transport Layer Security (DTLS) [RFC9147]. They also enable Packetization Layer Path MTU Discovery (PLPMTUD) over UDP, known as Datagram Packetization Layer Path Maximum Transmission Unit Discovery (DPLPMTUD) [RFC9869], providing a means for probe packet validation without affecting the user data plane, as well as providing explicit indication of the receiver transport reassembly size.

UDP originally assumed that such capabilities would be provided by the user or by a layer above UDP [RFC0768]. However, enough protocols have evolved to use UDP directly, so such an intermediate layer would be difficult to deploy for legacy applications. UDP options leverage the opportunity presented by the surplus area to enable these extensions within the UDP transport layer itself. Among the use cases where this approach could be of benefit are request- response protocols such as DNS over UDP [He24].

6. UDP Option Design Principles

UDP options have been designed based on the following core principles. Each is an observation about (preexisting) UDP [RFC0768] in the absence of these extensions that this document does not intend to change or a lesson learned from other protocol designs.

1. UDP is stateless; UDP options do not change that fact.

   The state required or maintained by the endpoints is intended to be managed either by the application or a layer/library on behalf of the application. Reassembly of fragments is the only limited exception where this document introduces a notion of state to UDP.

2. UDP is unidirectional; UDP options do not change that fact.

Responses to options are initiated by the application or a layer/library on behalf of the application. A mechanism that requires bidirectionality needs to be defined in a separate document.

3. UDP options have no length limit separate from that of the UDP packet itself.

   Past experience with other protocols confirms that static length limits will always need to be exceeded, e.g., as has been an issue with TCP options and IPv4 addresses. Each implementation can limit how long/many options there are, but a specification is more robust when it does not introduce such a limit.

4. UDP options are not intended to replace or replicate other protocols.

   This includes NTP, ICMP (notably echo), etc. UDP options are intended to introduce features useful for applications, not to either replace these other protocols nor instrument UDP to replace the need for network testing devices.

5. UDP options are a framework, not a protocol.

   Options can be defined in this initial document even when the details are not sufficient to specify a complete protocol. Uses of such options could then be described or supplemented in other documents. Examples herein include REQ/RES and TIME; in both cases, the option format is defined, but the protocol that uses these is specified elsewhere (REQ/RES for DPLPMTUD [RFC9869]) or left undefined (TIME).

6. The UDP option mechanism and UDP options themselves are intended to default to the same behavior experienced by a legacy receiver.

   By default, even when option checksums (OCS, APC), authentication, or decryption fail, all received packets (with the exception of UDP fragments) are passed (possibly with an empty data payload) to the user application. Options that do not modify user data are intended to (by default) result in the user data also being passed, even if, e.g., option checksums or authentication fails. It is always the user's or application's obligation to override this default behavior explicitly.

These principles are intended to enable the design and use of UDP options with minimal impact to legacy UDP endpoints, preferably none. UDP is -- and remains -- a minimal transport protocol. Additional capability is explicitly activated by user applications or libraries acting on their behalf.

Finally, UDP options do not attempt to match the number of zero- length UDP datagrams received by legacy and option-aware receivers from a source using UDP fragmentation (see Section 11.4). Legacy receivers interpret every UDP fragment as a zero-length packet (because they do not perform reassembly), but option-aware receivers would reassemble the packet as a non-zero-length packet. Zero-length UDP packets have been used as "liveness" indicators (see Section 5 of [RFC8085]), but such use is dangerous because they lack unique identifiers (the IPv6 base header has none, and the IPv4 ID field is deprecated for such use [RFC6994]).

7. The UDP Option Area

The UDP transport header includes demultiplexing and service identification (port numbers), an error detection checksum, and a field that indicates the UDP datagram length (including UDP header). The UDP Length field is typically redundant with the size of the maximum space available as a transport protocol payload, as determined by the IP header (see details in Section 18). The UDP option area is created when the UDP Length indicates a smaller transport payload than implied by the IP header.

For IPv4, the IP Total Length field indicates the total IP datagram length (including the IP header), and the size of the IP options is indicated in the IP header (in 4-byte words) as the "Internet Header Length" (IHL) [RFC0791], as shown in Figure 1. In exceptional cases, the Protocol field in IPv4 might not indicate UDP (i.e., 17), e.g., when intervening shim headers are present such as IP Security (IPsec) [RFC4301] or for IP Payload Compression (IPComp) [RFC3173].

The upper bound for UDP Length when Protocol = 17 is given by:

```
UDP_Length <= IPv4_Total_Length - IPv4_IHL * 4
```

If shim headers are present, this upper bound must be reduced by the sum of the lengths of shim headers that precede the UDP header.

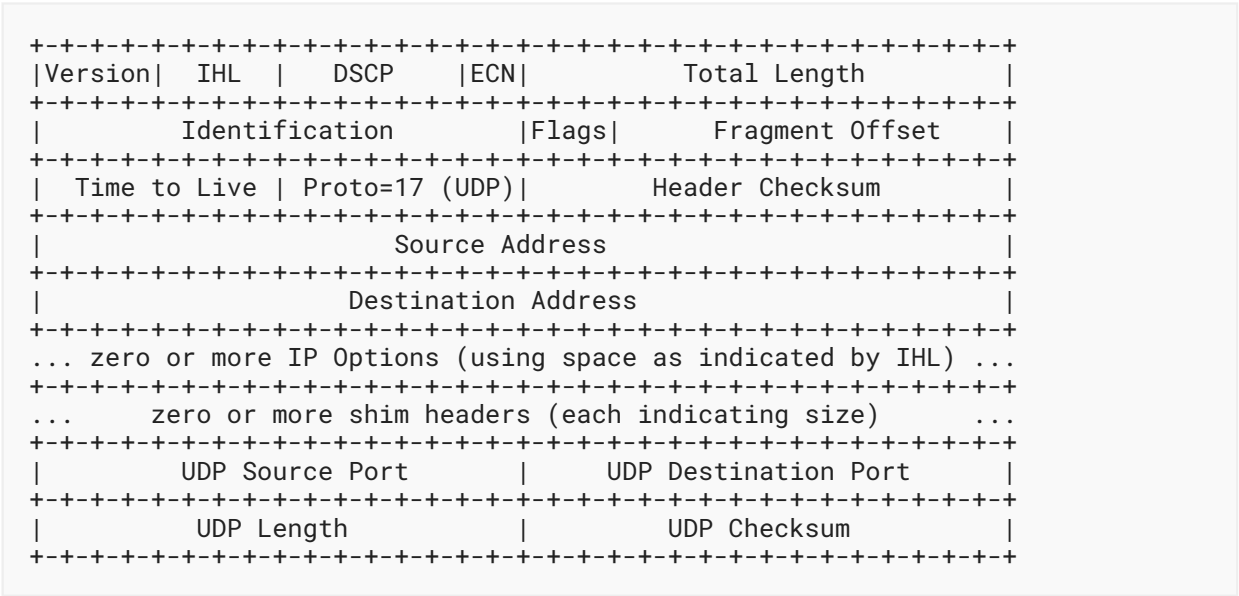

Figure 1: IPv4 Datagram with UDP Header

For IPv6, the IP Payload Length field indicates the transport payload after the base IPv6 header, which includes the IPv6 extension headers and space available for the transport protocol, as shown in Figure 2 [RFC8200]. Note that the Next Header field in IPv6 might not indicate UDP (i.e., 17), e.g., when intervening IP extension headers are present. For IPv6, the lengths of any additional IP extensions are indicated within each extension [RFC8200], so the upper bound for UDP Length is given by:

```
UDP_Length <= IPv6_Payload_Length - sum(extension header lengths)
```

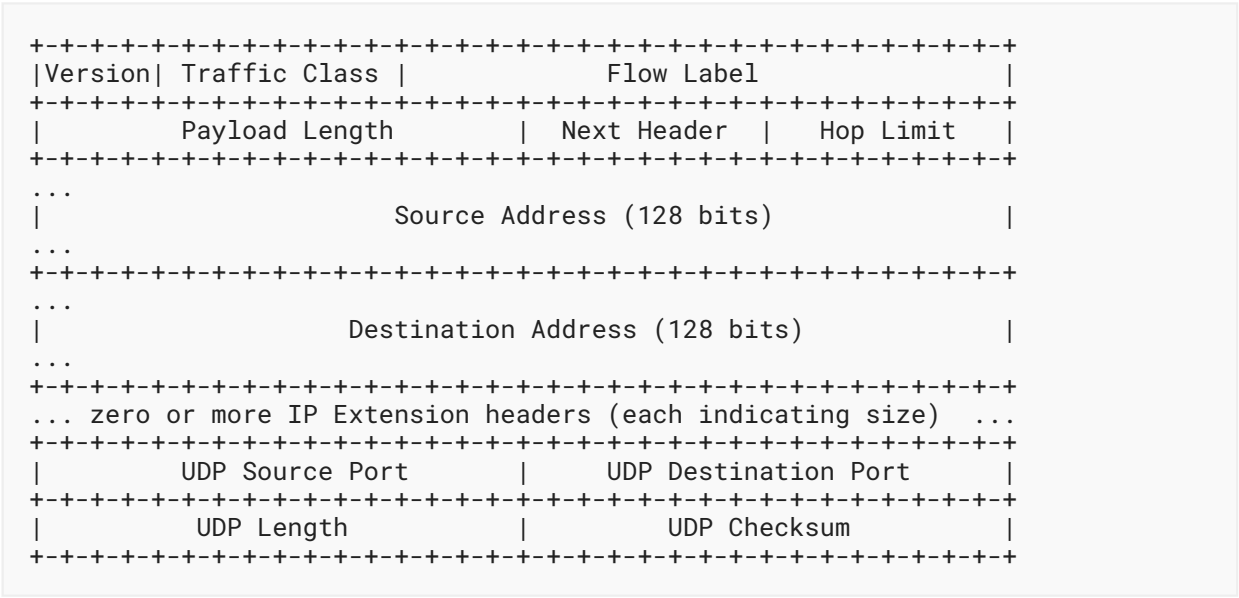

Figure 2: IPv6 Datagram with UDP Header

In both cases, the space available for the UDP packet is indicated by IP, either directly in the base header or by adding information in the shim headers or extensions. In either case, this document will refer to this available space as the "IP transport payload".

As a result of this redundancy, there is an opportunity to use the UDP Length field as a way to break up the IP transport payload into two areas -- that intended as UDP user data and an additional "surplus area" (as shown in Figure 3).

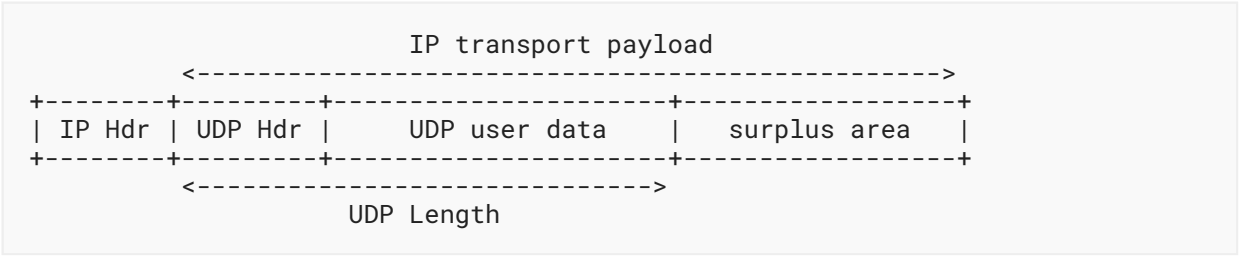

Figure 3: IP Transport Payload vs. UDP Length

In most cases, the IP transport payload and UDP Length point to the same location, indicating that there is no surplus area. This is not a requirement of UDP [RFC0768] (discussed further in Section 18). This document uses the surplus area for UDP options.

The surplus area can commence at any valid byte offset, i.e., it need not be 16-bit or 32-bit aligned. In effect, this document redefines the UDP Length field as a "trailer options offset".

8. The UDP Surplus Area Structure

UDP options use the entire surplus area, i.e., the contents of the IP payload after the last byte of the UDP payload. They commence with a 2-byte Option Checksum (OCS) field aligned to the first 2- byte boundary (relative to the start of the IP datagram) of that area, adding zeroes before OCS as needed for alignment. The UDP option area can be used with any UDP payload length (including zero, i.e., a UDP Length of 8), as long as there remains enough space for the aligned OCS and the options used.

>> UDP options **MAY** begin at any UDP length offset.

>> Option area bytes used for alignment before the OCS **MUST** be zero. If this is not the case, all options **MUST** be ignored and the surplus area silently discarded.

These alignment bytes, coupled with OCS as computed over the remainder of the surplus area, ensure that the one's complement sum of the surplus area is zero. OCS is half-word (2-byte) aligned to avoid the need for byte-swapping in its implementation.

The OCS contains an optional one's complement sum that detects errors in the surplus area, which is not otherwise covered by the UDP checksum, as detailed in Section 9.

The remainder of the surplus area consists of options, all except two of which are defined using a TLV (type, length, and optional value) syntax similar to that of TCP [RFC9293], as detailed in Section 10 (types No Operation (NOP) and End of Options List (EOL) have an implicit length of one byte). These options continue until the end of the surplus area or can end earlier using the EOL option, followed by zeroes (discussed further in Section 10).

9. The Option Checksum (OCS)

The Option Checksum (OCS) option is a conventional Internet checksum [RFC0791] that detects errors in the surplus area. The OCS option contains a 16-bit checksum that is aligned to the first 2-byte boundary, preceded by zeroes for padding (if needed), as shown in Figure 4.

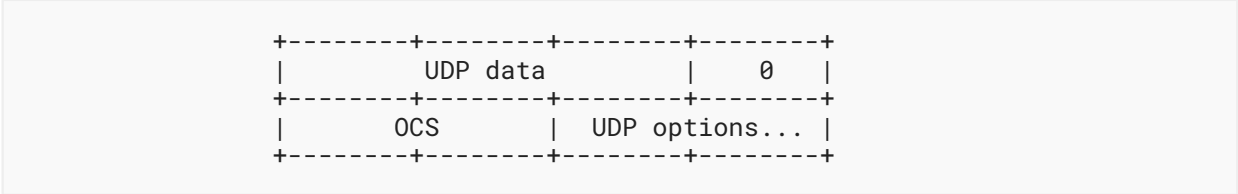

Figure 4: UDP OCS Format, Here Using One Zero Byte for Alignment

The OCS consists of a 16-bit Internet checksum [RFC1071], computed over the surplus area and including the length of the surplus area as an unsigned 16-bit value. The OCS protects the surplus area from errors in a similar way that the UDP checksum protects the UDP user data (when not zero).

The primary purpose of the OCS is to detect existing nonstandard (i.e., non-option) uses of that area and accidental errors. It is not intended to detect attacks, as discussed further in Section 25. OCS is not intended to prevent future nonstandard uses of the surplus area nor does it enable shared use with mechanisms that do not comply with UDP options.

The design enables traversal of errant middleboxes that incorrectly compute the UDP checksum over the entire IP payload [Fa18] [Zu20], rather than only the UDP header and UDP payload (as indicated by the UDP header length). Because the OCS is computed over the surplus area and its length and then inverted, the OCS effectively negates the effect that incorrectly including the surplus has on the UDP checksum. As a result, when OCS is non-zero, the UDP checksum is the same in either case.

>> The OCS **MUST** be non-zero when the UDP checksum is non-zero.

>> When the UDP checksum is zero, the OCS **MAY** be unused and is then indicated by a zero OCS value.

>> UDP option implementations **MUST** default to using the OCS (i.e., as a non-zero value); users overriding that default take the risk of not detecting nonstandard uses of the option area (of which there are none currently known).

Like the UDP checksum, the OCS is optional under certain circumstances and contains zero when not used. UDP checksums can be zero for IPv4 [RFC0791] and for IPv6 [RFC8200] when the UDP payload is already covered by another checksum, as might occur for tunnels [RFC6935]. The same exceptions apply to the OCS when used to detect bit errors; an additional exception occurs for its use in the UDP datagram prior to fragmentation or after reassembly (see Section 11.4).

The benefits are similar to allowing UDP checksums to be zero, but the risks differ. The OCS is additionally important to ensure packets with UDP options can traverse errant middleboxes [Zu20]. When the cost of computing an OCS is negligible, it is better to use the OCS to ensure such traversal. In cases where such traversal risks can safely be ignored, such as controlled environments, over paths where traversal is validated, or where upper layer protocols (applications, libraries, etc.) can adapt (by enabling the OCS when packet exchange fails), and when bit errors at the UDP layer would be detected by other layers (as with the UDP checksum), the OCS can be disabled, e.g., to conserve energy or processing resources or when performance can be improved. This is why zeroing the OCS is only safe when UDP checksum is also zero and why OCS might still be used in that case.

The OCS covers the surplus area as formatted for transmission and is processed immediately upon reception.

>> If the receiver validation of the OCS fails, all options **MUST** be ignored and the surplus area silently discarded.

>> UDP user data that is validated by a correct UDP checksum **MUST** by default be delivered to the application layer, even if the OCS fails, unless the endpoints have negotiated otherwise for this UDP packet's socket pair.

When not used (i.e., containing zero), the OCS is assumed to be "correct" for the purpose of accepting UDP datagrams at a receiver (see Section 14).

10. UDP Options

UDP options are a minimum of two bytes in length as shown in Figure 5, except only the one-byte options No Operation (NOP) and End of Options List (EOL) described below.

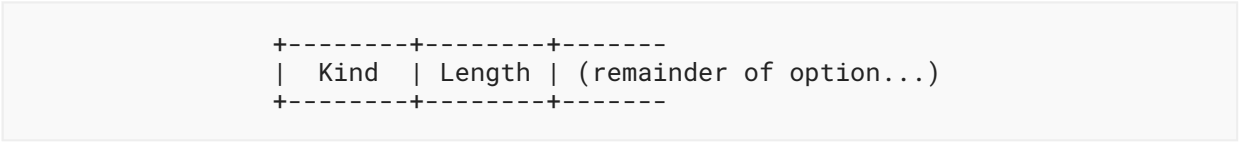

Figure 5: UDP Option Default Format

The Kind field is always one byte and is named after the corresponding TCP field (though other protocols refer to this as "Type"). The Length field, which indicates the length in bytes of the entire option, including Kind and Length, is one byte for all lengths below 255 (including the Kind and Length bytes). A Length of 255 indicates use of the UDP option extended format shown in Figure 6. The Extended Length field is a 16-bit field in network standard byte order. The length of the option refers to its Length field or Extended Length field, whichever is applicable.

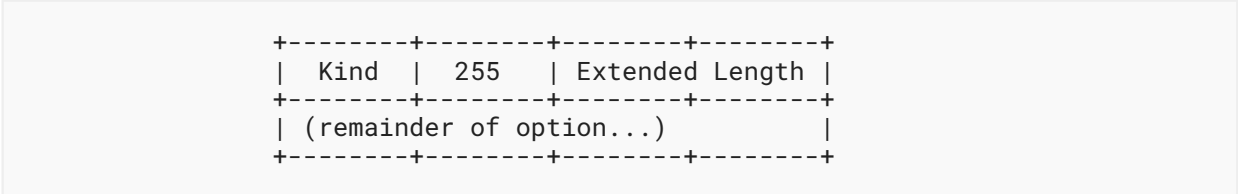

Figure 6: UDP Option Extended Format

>> The UDP length **MUST** be at least as large as the UDP header (8) and no larger than the IP transport payload. Datagrams with length values outside this range **MUST** be silently dropped as invalid and logged.

>> All logging **SHOULD** be rate limited. Excess logging events can be coalesced and reported as a count or can be silently dropped if needed to avoid resource overloading.

>> Option Lengths (or Extended Lengths, where applicable) smaller than the minimum for the corresponding Kind **MUST** be treated as an error. Such errors call into question the remainder of the surplus area and thus **MUST** result in all UDP options being silently discarded.

>> Any UDP option other than NOP or EOL whose length is 254 or less **MUST** use the UDP option default format shown in Figure 5. NOP and EOL never use either length format.

>> Any UDP option whose length is larger than 254 **MUST** use the UDP option extended format shown in Figure 6.

>> For compactness, UDP options **SHOULD** use the smallest option format possible.

>> UDP options **MUST** be interpreted in the order in which they occur in the surplus area or, in the case of UDP fragments, in the order in which they appear in the UDP fragment option area (see Section 11.4).

The following UDP options are currently defined:

Kind	Length	Meaning
0*	-	End of Options List (EOL)
1*	-	No Operation (NOP)
2*	6	Additional Payload Checksum (APC)
3*	10/12	Fragmentation (FRAG)
4*	4	Maximum Datagram Size (MDS)
5*	5	Maximum Reassembled Datagram Size (MRDS)
6*	6	Request (REQ)

Kind	Length	Meaning
7*	6	Response (RES)
8	10	Timestamps (TIME)
9	(varies)	RESERVED for Authentication (AUTH)
10-126	(varies)	Unassigned (assignable by IANA)
127	(varies)	RFC3692-style experiments (EXP)
128-191		Reserved
192	(varies)	Reserved for Compression (UCMP)
193	(varies)	Reserved for Encryption (UENC)
194-253		Unassigned-UNSAFE (assignable by IANA)
254	(varies)	RFC3692-style experiments (UEXP)
255		Reserved-UNSAFE

Table 1

Options indicated by Kind values in the range 0..191 are known as SAFE options because they do not interfere with use of that data by legacy endpoints or when the option is unsupported. Options indicated by Kind values in the range 192..255 are known as UNSAFE options because they might interfere with use by legacy receiving endpoints (e.g., an option that alters the UDP data payload).

UNSAFE option nicknames are expected to begin with capital "U", which needs to be avoided for SAFE option nicknames (see Section 26). RESERVED and RESERVED-UNSAFE are not assignable by IANA and not otherwise defined at this time. The AUTH, UCMP, and UENC reservations are intended for all future options supporting authentication, compression, and encryption, respectively, and remain reserved until assigned for those uses.

Although the FRAG option modifies the original UDP payload contents (i.e., is UNSAFE with respect to the original UDP payload), it is used only in subsequent fragments with zero-length UDP user data payloads, thus is SAFE in actual use, as discussed further in Section 11.4.

These options are defined in the following subsections. Options 0 and 1 use the same values as for TCP.

>> An endpoint supporting UDP options **MUST** support those marked with an "*" above: EOL, NOP, APC, FRAG, MDS, MRDS, REQ, and RES. This includes both recognizing and being able to generate these options if configured to do so. These are called "must-support" options.

The set of must-support options is defined herein. New options are not eligible for this designation.

>> All other SAFE options (without an "*") **MAY** be implemented, and their use **SHOULD** be determined either out-of-band or negotiated, notably if needed to detect when options are silently ignored by legacy receivers.

>> Receivers supporting UDP options **MUST** silently ignore unknown or malformed SAFE options (i.e., in the same way a legacy receiver would ignore all UDP options). An option is malformed when its length does not indicate (one of) the value(s) stated in the option's specification. A malformed FRAG option is an exception to this rule; it **SHALL** be treated as an unsupported UNSAFE option.

>> Options with inherently invalid Length field values, i.e., those that indicate underruns of the option itself or overruns of the surplus area (pointing past the end of the IP payload), **MUST** be treated as an indication of a malformed surplus area, and all options **MUST** silently be discarded.

Receivers cannot generally treat unexpected option lengths as invalid, as this would unnecessarily limit future revision of options (e.g., defining a new APC that is defined by having a different length).

>> When UNSAFE options are present, the UDP user data **MUST** be empty, and any transport payload **MUST** be contained in a FRAG option (see Section 11.4). Recall that such options may alter the transport payload or signal a change in what its contents represent. This restriction ensures their safe use in environments that might include legacy receivers (see Section 12), because the transport payload occurs inside the FRAG option area and is silently discarded by legacy receivers.

>> Receivers supporting UDP options that receive unsupported options in the UNSAFE range **MUST** terminate all option processing and **MUST** silently drop all UDP options in that datagram. See Section 12 for further discussion of UNSAFE options.

>> Other than FRAG, NOP, EXP, and UEXP, each option **SHOULD NOT** occur more than once in a single UDP datagram. If an option other than these four occurs more than once, a receiver **MUST** interpret only the first instance of that option and **MUST** ignore later instances. Section 25 provides additional advice for Denial of Service (DoS) issues that involve large numbers of options, whether valid, unknown, or repeating.

>> NOP **MAY** occur multiple times, either in succession or between other options, for option alignment. Additional repetition constraints are indicated in Section 11.2.

>> If FRAG occurs more than once, the options area **MUST** be considered malformed and **MUST NOT** be processed.

>> EXP and UEXP **MAY** occur more than once but **SHOULD NOT** occur more than once using the same Experimental ID (ExID) (see Sections 11.10 and 12.3).

>> Options other than OCS, AUTH, and UENC **MUST NOT** include fields whose values depend on the contents of the surplus area.

AUTH and UENC are always computed as if their hash and the OCS are zero; the OCS is always computed as if its contents are zero and after the AUTH or UENC hash has been computed.

>> Future options **MUST NOT** be defined as having an option field value dependent on the content or presence of other options or on the remaining contents of the surplus area, i.e., the area after the last option (presumably EOL).

If future options were to depend on the contents or presence of other options, interactions between those values, the OCS, and the AUTH and UENC options could be unpredictable. This does not prohibit options that modify later options (in order of appearance within a packet), such as would typically be the case for compression (UCMP).

Note that there is no need to reserve area after the last UDP option for future uses, because any such use can be supported by defining a new UDP option over that area instead. Using an option for this purpose is safer than treating the region as an exception, because its use can be verified based on option Kind and Length.

>> AUTH and UENC **MUST NOT** be used concurrently.

AUTH and UENC are never used together because UENC would serve both purposes.

>> "Must-support" options other than NOP and EOL **MUST** be placed by the transmitter before other SAFE UDP options. A receiver **MAY** drop all UDP options if this ordering is not honored. Such events **MAY** be logged for diagnostic purposes.

The requirement that must-support options come before others is intended to allow for endpoints to implement DoS protection, as discussed further in Section 25.

11. SAFE UDP Options

SAFE UDP options can be silently ignored by legacy receivers without affecting the meaning of the UDP user data. They stand in contrast to UNSAFE options, which modify UDP user data in ways that render it unusable by legacy receivers (Section 12). The following subsections describe SAFE options defined in this document.

11.1. End of Options List (EOL)

The End of Options List (EOL, Kind=0) option indicates that there are no more options. It is used to indicate the end of the list of options without needing to use NOP options (see the following section) as padding to fill all available option space.

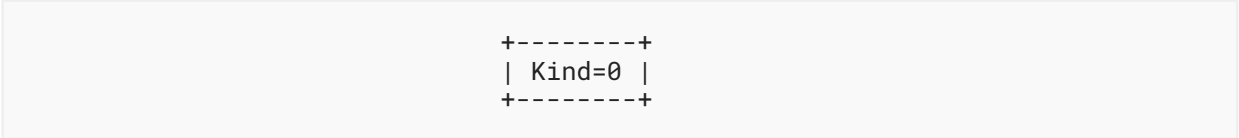

Figure 7: UDP EOL Option Format

>> When the UDP options do not consume the entire surplus area or the options area of a UDP fragment, the last non-NOP option **MUST** be EOL.

>> NOPs **SHOULD NOT** be used as padding before the EOL option. As a one-byte option, EOL need not be otherwise aligned.

>> All bytes after EOL in the surplus area or the options area of a UDP fragment **MUST** be set to zero on transmit.

>> Bytes after EOL in the surplus area or the options area of a UDP fragment **MAY** be checked as being zero on receipt but **MUST NOT** be otherwise processed (except for OCS calculation, which zeros would not affect) and **MUST NOT** be passed to the user.

>> If a receiver elects to check the bytes following EOL and finds that they are not all set to zero, it **MUST** silently discard the options area. In this case, the UDP user data **MUST** be delivered to the application layer, unless the socket has been explicitly configured to do otherwise, as decided by the upper layer or negotiated with the other endpoint.

Requiring the post-option surplus area to be zero prevents side- channel uses of this area, instead requiring that all use of the surplus area be UDP options supported by both endpoints. It is useful to allow this area to be used for zero padding to increase the UDP datagram length without affecting the UDP user data length, e.g., for UDP DPLPMTUD (Section 4.1 of [RFC9869]).

11.2. No Operation (NOP)

The No Operation (NOP, Kind=1) option is a one-byte placeholder, intended to be used as padding, e.g., to align multi-byte options along 16-bit, 32-bit, or 64-bit boundaries.

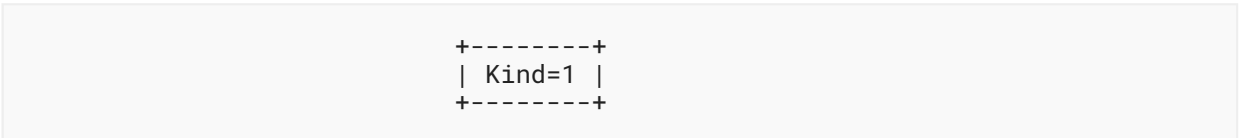

Figure 8: UDP NOP Option Format

>> UDP packets **SHOULD NOT** use more than seven consecutive NOPs, i.e., to support alignment up to 8-byte boundaries. UDP packets **SHOULD NOT** use NOPs at the end of the options area as a substitute for EOL followed by zero-fill. NOPs are intended to assist with alignment, not as other padding or fill.

>> Receivers persistently experiencing packets with more than seven consecutive NOPs **SHOULD** log such events, at least occasionally, as a potential DoS indicator.

NOPs are not reported to the user, whether used per-datagram or per- fragment (as defined in Section 11.4).

This issue is discussed further in Section 25.

11.3. Additional Payload Checksum (APC)

The Additional Payload Checksum (APC, Kind=2) option provides a stronger supplement to the checksum in the UDP header, using a 32- bit Cyclic Redundancy Check (CRC) of the conventional UDP user data payload only (excluding the IP pseudoheader, UDP header, and surplus area). It is not an alternative to the UDP checksum because it does not cover the IP pseudoheader or UDP header, and it is not a supplement to the OCS because the latter covers the surplus area only. Its purpose is to detect user data errors that the UDP checksum might not detect.

A CRC32c has been chosen because of its ubiquity and use in other Internet protocols, including Internet Small Computer System Interface (iSCSI) [RFC3385] and SCTP. The option contains the CRC32c in network standard byte order, as used for iSCSI.

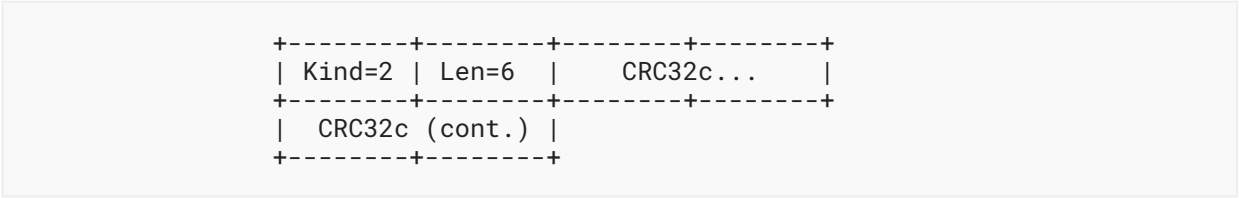

Figure 9: UDP APC Option Format

When present, the APC always contains a valid CRC checksum. There are no reserved values, including the value zero. A CRC value of zero is a potentially valid checksum. As such, it does not indicate that the APC is not used; instead, the option would simply not be included if that were the desired effect.

The APC does not protect the UDP pseudoheader; only the current UDP checksum provides that protection (when used). The APC cannot provide that protection because it would need to be updated whenever the UDP pseudoheader changed, e.g., during NAT address and port translation (see [RFC1141]).

>> UDP packets with incorrect APC checksums **SHOULD** be passed to the application with an indication of APC failure. This is the default behavior for APC.

>> Like all SAFE UDP options, the APC **MUST** be silently ignored when failing, unless the receiver has been explicitly configured to do otherwise.

Although all UDP option-aware endpoints support the APC (being in the required set), this silently ignored behavior ensures that option- aware receivers operate the same as legacy receivers unless overridden. Another reason is because the APC check could fail even where the user data has not been corrupted, such as when its contents have been intentionally overwritten, e.g., by a middlebox to update embedded port numbers or IP addresses. Such overwrites could be intentional and not widely known; defaulting to silent ignore ensures that option-aware endpoints do not change how users or applications operate unless explicitly directed to do otherwise.

>> UDP packets with unrecognized APC lengths **MUST** receive the same treatment as UDP packets with incorrect APC checksums.

Ensuring that unrecognized APC lengths are treated as incorrect checksums enables future variants of APC to be treated like APC.

The APC is reported to the user and useful only per-datagram, because fragments have no UDP user data.

11.4. Fragmentation (FRAG)

The Fragmentation (FRAG, Kind=3) option supports UDP fragmentation and reassembly, which can be used to transfer UDP messages larger than allowed by the IP receive MTU (Effective MTU for Receiving (EMTU_R) [RFC1122]). FRAG includes a copy of the same UDP transport ports in each fragment, enabling them to traverse stateless Network Address (and port) Translation (NAT) devices, in contrast to the behavior of IP fragments [RFC4787]. FRAG is typically used with the UDP MDS and MRDS options to enable more efficient use of large messages, both at the UDP and IP layers. The design of FRAG is similar to that of the IPv6 Fragmentation Header [RFC8200], except that the UDP variant uses a 16-bit Offset measured in bytes, rather than IPv6's 13-bit Fragment Offset measured in 8-byte units. This UDP variant avoids creating reserved fields.

The FRAG header also enables use of options that modify the contents of the UDP payload, such as encryption (UENC, see Section 12.2). Like FRAG, such options would not be safely used on UDP payloads because they would be misinterpreted by legacy receivers. FRAG allows use of these options, either on fragments or on a whole, unfragmented message (i.e., an "atomic" fragment at the UDP layer, similar to atomic IP datagrams [RFC6864]). This is safe because FRAG hides the payload from legacy receivers by placing it within the surplus area.

>> When FRAG is present, it **SHOULD** come as early as possible in the UDP options list.

When present, placing FRAG first can simplify some implementations, notably those using hardware acceleration that assume a fixed location for the FRAG option. However, there are cases where FRAG cannot occur first, such as when combined with per-fragment UENC or UCMP. In those cases, encryption or compression (or both) would precede FRAG when they also encrypt or compress the fragment option itself. Additional cases could include recoding, such as could be used to support Forward Error Correction (FEC) over a group of fragments. FRAG not being first might result in software (so-called "slow path") option processing or might also be accommodated via a small set of known cases.

>> When FRAG is present, the UDP user data **MUST** be empty. If the user data is not empty, all UDP options **MUST** be silently ignored and the user data received sent to the user.

Legacy receivers interpret FRAG messages as zero-length user data UDP packets (i.e., UDP Length field is 8, the length of just the UDP header), which would not affect the receiver unless the presence of the UDP packet itself were a signal (see Section 5 of [RFC8085]). In this manner, the FRAG option also helps hide UNSAFE options so they can be used more safely in the presence of legacy receivers.

The FRAG option has two formats: non-terminal fragments use the shorter variant (Figure 10) and terminal fragments use the longer (Figure 11). The latter includes stand-alone fragments, i.e., when data is contained in the FRAG option but reassembly is not required.

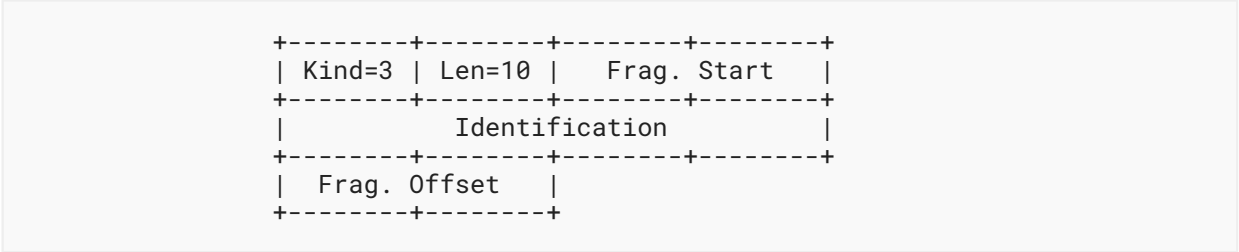

Figure 10: UDP Non-Terminal FRAG Option Format

Most fields are common to both FRAG option formats. The option Len field indicates whether there are more fragments (Len=10) or no more fragments (Len=12).

The Frag. Start field indicates the location of the beginning of the fragment data, measured from the beginning of the UDP header of the fragment. The fragment data follows the remainder of the UDP options and continues to the end of the IP datagram (i.e., the end of the surplus area). Those options (i.e., any that precede or follow the FRAG option) are applied to this UDP fragment.

The Frag. Offset field indicates the location of this fragment relative to the original UDP datagram (prior to fragmentation or after reassembly), measured from the start of the original UDP datagram's header.

The Identification field is a 32-bit value that, when used in combination with the IP source address, UDP source port, IP destination address, and UDP destination port, uniquely identifies the original UDP datagram.

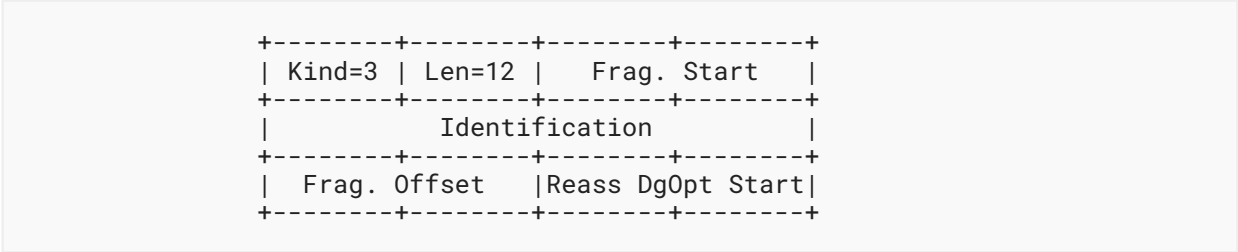

Figure 11: UDP Non-Terminal FRAG Option Format

The terminal FRAG option format adds a Reassembled Datagram Option Start (RDOS) pointer, measured from the start of the original UDP datagram header, indicating the end of the reassembled data and the start of the surplus area within the original UDP datagram. UDP options that apply to the reassembled datagram are contained in the reassembled surplus area, as indicated by RDOS. UDP options that occur within the fragment are processed on the fragment itself. This allows either pre-reassembly or post-reassembly UDP option effects, such as using UENC on each fragment while also using TIME on the reassembled datagram for round-trip latency measurements.

An example showing the relationship between UDP fragments and the original UDP datagram is provided in Figure 12. In this example, the trailer containing per-datagram options resides entirely within the terminal fragment, but this need not always be the case.

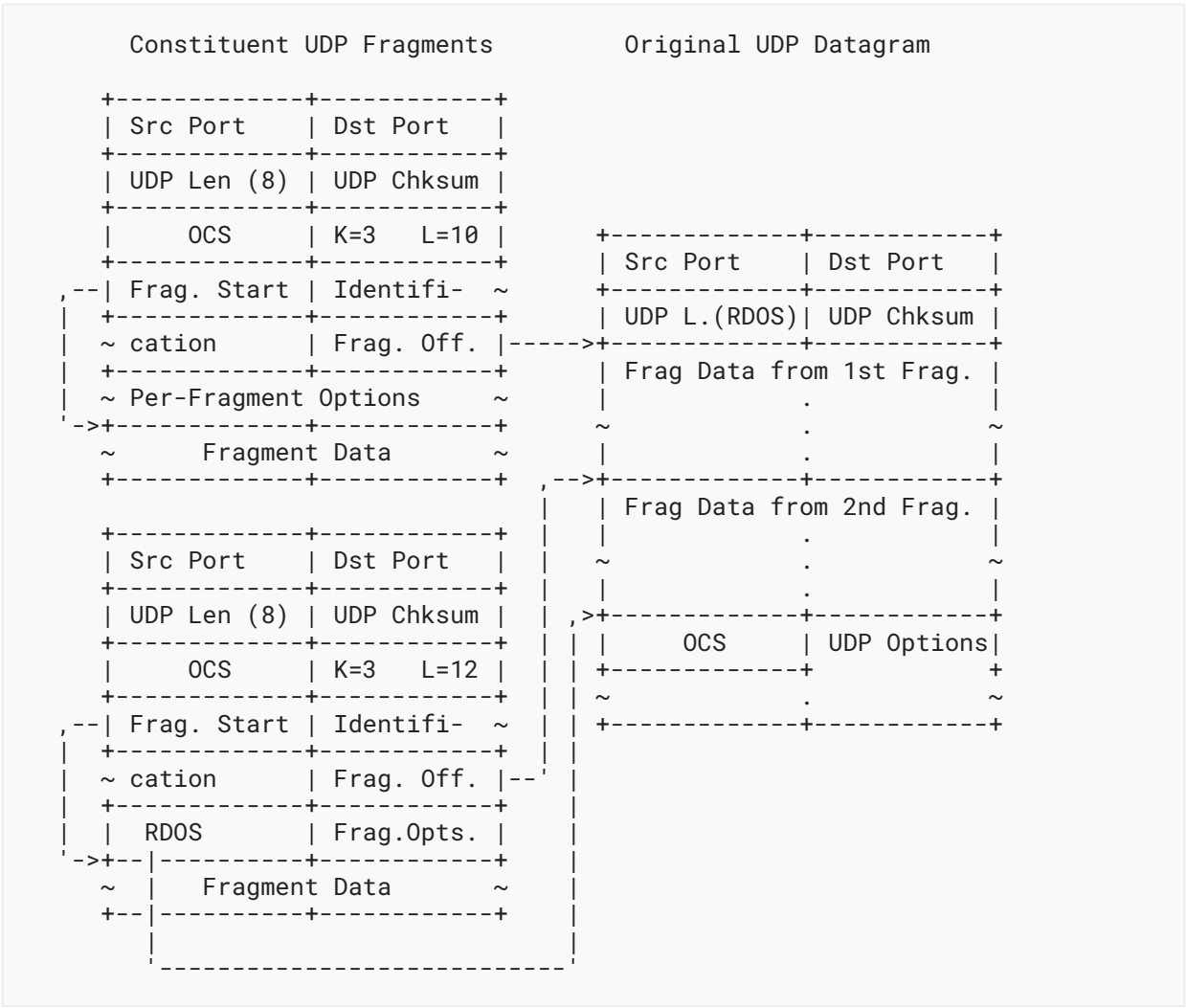

Figure 12: UDP Fragments and Original UDP Datagram

The FRAG option does not need a "more fragments" bit (as used by IP fragmentation) because it provides the same indication by using the longer, 12-byte variant, as shown in Figure 11.

>> The FRAG option **MAY** be used on a single fragment; in which case, the Frag. Offset would be zero and the option would have the 12-byte format.

>> Endpoints supporting UDP options **MUST** be capable of fragmenting and reassembling at least two fragments, each of a size that will fit within the standard Ethernet MTU of 1,500 bytes. For further details, please see Section 11.6.

Use of the single fragment variant can be helpful in supporting use of UNSAFE options without undesirable impact to receivers that do not support either UDP options or the specific UNSAFE options.

During fragmentation, the UDP header checksum of each fragment remains constant. It does not depend on the fragment data (which appears in the surplus area) because all fragments have a zero- length user data field.

>> The Identification field is a 32-bit value that **MUST** be unique over the expected fragment reassembly timeout.

>> The Identification field **SHOULD** be generated in a manner similar to that of the IPv6 Fragment ID [RFC8200].

>> UDP fragments **MUST NOT** overlap.

>> Similar to IPv6 reassembly [RFC8200], if any of the fragments being reassembled overlap with any other fragments being reassembled for the same UDP packet, reassembly of that UDP packet **MUST** be abandoned and all the fragments that have been received for that UDP packet **MUST** be discarded, and no ICMP error messages are to be sent in this case (to avoid a potential DoS attack turning into an ICMP storm in the reverse direction).

>> Note that fragments might be duplicated in the network. Instead of treating these exact duplicate fragments as overlapping fragments, an implementation **MAY** choose to detect this case and drop exact duplicate fragments while keeping the other fragments belonging to the same UDP packet.

UDP fragmentation relies on a fragment expiration timer, which can be preset or could use a value computed using the UDP Timestamp option.

>> The default UDP reassembly expiration timeout **SHOULD** be no more than 2 minutes.

>> UDP reassembly expiration **MUST NOT** generate an ICMP error. Such events are not an IP error and can be addressed by the user/application layer if desired.

>> UDP reassembly space **SHOULD** be limited to reduce the impact of DoS attacks on resource use.

>> UDP reassembly space limits **SHOULD NOT** be computed as a shared resource across multiple sockets, to avoid cross-socket pair DoS attacks.

>> Individual UDP fragments **MUST NOT** be forwarded to the user. The reassembled datagram is received only after complete reassembly, checksum validation, and continued processing of the remaining UDP options.

Per-fragment UDP options, if used in addition to FRAG, occur before the fragment data. They typically occur after the FRAG option, except where they modify the FRAG option itself (e.g., UENC or UCMP). Per-fragment options are processed before the fragment is included in the reassembled datagram. Such options can be useful to protect the reassembly process itself, e.g., to prevent the reassembly cache from being polluted (using AUTH or UENC).

>> Fragments of a single datagram **MAY** use different sets of options. It is expected to be computationally expensive to validate uniformity across all fragments, and there could be legitimate reasons for including options in a fragment but not all fragments (e.g., MDS and MRDS).

If an option is used per-fragment but defined as not usable per- fragment, it is treated the same as any other unknown option.

Per-datagram UDP options, if used, reside in the surplus area of the original UDP datagram. Processing of those options occurs after reassembly is complete. This enables the safe use of UNSAFE options, which are required to result in discarding the entire UDP datagram if they are unknown to the receiver or otherwise fail (see Section 12).

In general, UDP packets are fragmented as follows:

1. Create a UDP packet with data and UDP options. This is the original UDP datagram, which we will call "D". The UDP options follow the UDP user data and occur in the surplus area, just as in an unfragmented UDP datagram with UDP options.

   >> UDP options for the original packet **MUST** be fully prepared before the rest of the fragmentation steps that follow here.

   >> The UDP checksum of the original packet **SHOULD** be set to zero because it is never transmitted. Equivalent protection is provided if each fragment has a non-zero OCS value, as will be the case if each fragment's UDP checksum is non-zero. Similarly, the OCS value of the original packet **SHOULD** be zero if each fragment will have a non-zero OCS value, as will be the case if each fragment's UDP checksum is non-zero.
2. Identify the desired fragment size, which we will call "S". This value is calculated to take into account the path MTU (if known) and to allow space for per-fragment options.
3. Fragment "D" into chunks of size no larger than "S"-12 each (10 for the non-terminal FRAG option and 2 for OCS), with one final chunk no larger than "S"-14 (12 for the terminal FRAG option and 2 for OCS). Note that all the per-datagram options in step #1 need not be limited to the terminal fragment, i.e., the RDOS pointer can indicate the start of the original surplus area anywhere in the reassembled datagram.
4. For each chunk of "D" in step #3, create a UDP packet with no user data (UDP Length=8) followed by the word-aligned OCS, the FRAG option, and any additional per-fragment UDP options, followed by the FRAG data chunk.
5. Complete the processing associated with creating these additional per-fragment UDP options for each fragment.

Receivers reverse the above sequence. They process all received options in each fragment. When the FRAG option is encountered, the FRAG data is used in reassembly. After all fragments are received, the entire UDP packet is processed with any trailing UDP options applying to the reassembled user data.

>> Reassembly failures at the receiver result in silent discard of any per-fragment options and fragment contents, and such failures **SHOULD NOT** generate zero-length frames to the user.

>> Finally, because fragmentation processing can be expensive, the FRAG option **SHOULD** be avoided unless the original datagram requires fragmentation or it is needed for "safe" use of UNSAFE options.

>> The FRAG option **MAY** also be used to provide limited support for UDP options in systems that have access to only the initial portion of the data in incoming or outgoing packets, as such systems could potentially access per-fragment options. Such packets would, of course, be silently ignored by legacy receivers that do not support UDP options.

The presence of the FRAG option is not reported to the user.

11.5. Maximum Datagram Size (MDS)

The Maximum Datagram Size (MDS, Kind=4) option is a 16-bit hint of the largest UDP packet or UDP fragment that an endpoint believes can be received without use of IP fragmentation. It helps UDP applications limit the largest UDP packet that can be sent without UDP fragmentation and helps UDP fragmentation determine the largest UDP fragment to send -- in both cases, to avoid IP fragmentation.

As with the TCP Maximum Segment Size (MSS) option [RFC9293], the size indicated is the IP layer MTU decreased by the fixed IP and UDP headers only [RFC9293]. The space needed for IP and UDP options needs to be adjusted by the sender when using the value indicated. The value transmitted is based on EMTU_R, the largest IP datagram that can be received (i.e., reassembled at the receiver) [RFC1122]. However, as with TCP, this value is only a hint at what the receiver believes, as when used with PLPMTUD at the UDP layer, as discussed later in this section.

>> MDS does not indicate a known path MTU and thus **MUST NOT** be used to limit transmissions.

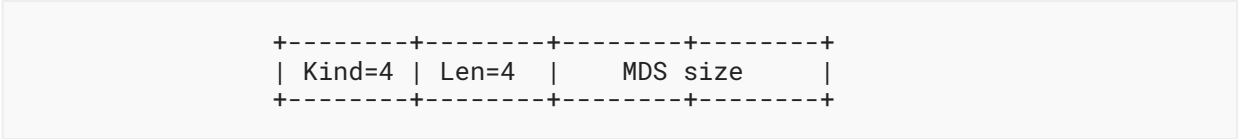

Figure 13: UDP MDS Option Format

>> The UDP MDS option **MAY** be used as a hint for path MTU discovery [RFC1191] [RFC8201], but this could be difficult because of known issues with ICMP blocking [RFC2923] as well as UDP lacking automatic retransmission.

MDS is more likely to be useful when coupled with IP source fragmentation or UDP fragmentation to limit the largest reassembled UDP message as indicated by MRDS (see Section 11.6), e.g., when EMTU_R is larger than the required minimums (576 for IPv4 [RFC0791] and 1500 for IPv6 [RFC8200]).

>> MDS can be used with DPLPMTUD [RFC8899] to provide a hint to the Packetization Layer Path MTU (PLPMTU) value, though it **MUST NOT** prohibit transmission of larger UDP packets used as DPLPMTUD probes.

MDS is reported to the user, whether used per-datagram or per- fragment (as defined in Section 11.4). When used per-fragment, the reported value is the minimum of the MDS values received per- fragment.

11.6. Maximum Reassembled Datagram Size (MRDS)

The Maximum Reassembled Datagram Size (MRDS, Kind=5) option is a 16- bit indicator of the largest reassembled UDP datagram that can be received, including the UDP header and any per-datagram UDP options, accompanied by an 8-bit indication of how many UDP fragments can be reassembled. MRDS size is the UDP equivalent of IP's EMTU_R, but the two are not related [RFC1122]. Using the FRAG option (Section 11.4), UDP packets can be transmitted as transport fragments, each in their own (presumably not fragmented) IP datagram, and be reassembled at the UDP layer. MRDS segs is the number of UDP fragments that can be reassembled.

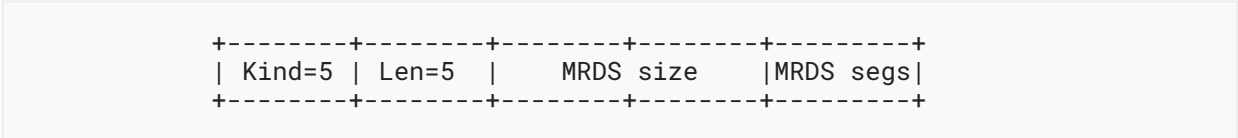

Figure 14: UDP MRDS Option Format

>> Endpoints supporting UDP options **MUST** support a local MRDS size of at least 2,926 bytes for IPv4 and 2,886 bytes for IPv6. Support for larger values is encouraged.

>> Endpoints supporting UDP options **MUST** support a local MRDS segs value of at least 2. Support for larger values is encouraged.

These parameters plus the Path MTU (PMTU) allow a sender to compute the size of the largest pre-fragmentation UDP packet that a receiver will guarantee to accept. Suppose that MMS_S is the PMTU less the size of the IP header and the UDP header, i.e., the maximum UDP message size that can be successfully sent in a single UDP datagram if there are no IP options or extension headers and no UDP per-fragment options.

Then, the size of the largest pre-fragmentation UDP packet that the receiver will guarantee to accept is the smaller of the MRDS size and

(MMS_S - 12) * (MRDS segs) - 2 - (Total Per-Frag IP/UDP Options) + 8

where Total Per-Frag IP/UDP Options includes the size of all IP options and extension headers and all per-fragment UDP options, except for OCS and FRAG, that are in the sequence of UDP fragments.

>> If no MRDS option has been received, a sender **MUST** assume that MRDS size is 2,926 bytes for IPv4 and 2,886 bytes for IPv6 and that MRDS segs is 2, i.e., the minimum values allowed.

MRDS is reported to the user, whether used per-datagram or per- fragment (as defined in Section 11.4). When used per-fragment, the reported value is the minimum of the MRDS values received per- fragment.

11.7. Echo Request (REQ) and Echo Response (RES)

The echo Request (REQ, Kind=6) and echo Response (RES, Kind=7) options provides UDP packet-level acknowledgments as a capability for use by upper layer protocols, e.g., user applications, libraries, operating systems, etc. Both the REQ and RES are under the control of these upper layers, i.e., UDP option support described in this document never automatically responds to a REQ with a RES. Instead, the REQ is delivered to the upper layer, which decides whether and when to issue a RES.

One such use is described as part of DPLPMTUD [RFC9869]. This use case is described as part of UDP options but is logically considered to be a capability of an upper layer that uses UDP options. The options both have the format indicated in Figure 15, in which the token has no internal structure or meaning.

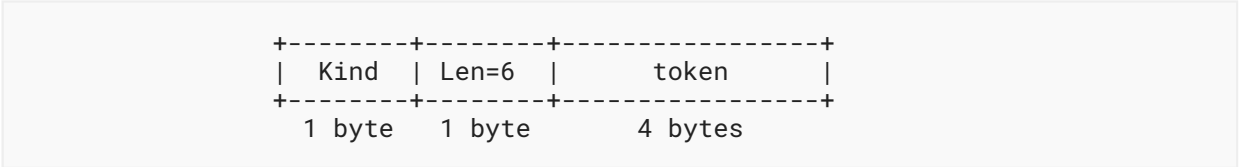

Figure 15: UDP REQ and RES Options Format

>> As advice to upper layer protocol/library designers, when supporting REQ/RES and responding with a RES, the upper layer **SHOULD** respond with the most recently received REQ token.

>> If the implementation includes a layer/library that produces and consumes REQ/RES on behalf of the user/application, then that layer **MUST** be disabled by default; in which case, REQ/RES are simply sent upon request by the user/application and passed to it when received, as with most other UDP options.

For example, an application needs to explicitly enable the generation of a RES response by DPLPMTUD when using UDP Options [RFC9869].

>> The token transmitted in a RES option **MUST** be a token received in a REQ option by the transmitter. This ensures that the response is to a received request.

REQ and RES option kinds each appear at most once in each UDP packet, as with most other options. A single packet can include both options, though they would be otherwise unrelated to each other. Note also that the FRAG option is not used when sending DPLPMTUD probes to determine a PLPMTU [RFC9869].

REQ and RES are reported to the user, whether used per-datagram or per-fragment (as defined in Section 11.4). When used per-fragment, the reported value indicates the most recently received token.

11.8. Timestamps (TIME)

Timestamps are provided as a capability to be used by applications and other upper layer protocols. They are based on a notion of time as a monotonically non-decreasing unsigned integer, with wraparound. They are defined the same way as TCP Protection Against Wrapped Sequence (PAWS) numbers, i.e., "without any connection to [real-world, classical physics wall-clock] time" [RFC7323]. They are quite similar to the behavior of relativistic time or the individual scalars of Lamport clocks [La78]. However, if desired, they can correspond to real-world time, e.g., as used for round-trip time (RTT) estimation. This option makes no assertions as to which is the case; the decision is up to the application layer using this option.

The Timestamp (TIME, Kind=8) option exchanges two four-byte unsigned timestamp fields. It serves a similar purpose to TCP's TS option [RFC7323], enabling UDP to estimate the RTT between hosts. For UDP, this RTT can be useful for establishing UDP fragment reassembly timeouts or transport-layer rate limiting [RFC8085].

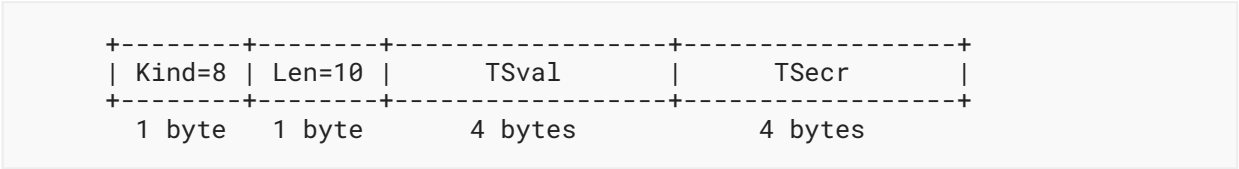

Figure 16: UDP TIME Option Format

TS Value (TSval) and TS Echo Reply (TSecr) are used in a similar manner to the TCP TS option [RFC7323]. On transmitted UDP packets using the option, TSval is always set based on the local "time" value. Received TSval and TSecr values are provided to the application, which can pass the TSval value to be used as TSecr on UDP messages sent in response (i.e., to echo the received TSval). A received TSecr of zero indicates that the TSval was not echoed by the transmitter, i.e., from a previously received UDP packet.

>> TIME **MAY** use an RTT estimate based on non-zero Timestamp values as a hint for fragmentation reassembly, rate limiting, or other mechanisms that benefit from such an estimate.

>> An application **MAY** use TIME to compute this RTT estimate for further use by the user.

UDP timestamps are modeled after TCP timestamps and have similar expectations. In particular, they are expected to follow these guidelines:

- Values are monotonic and non-decreasing except for anticipated number-space rollover events.
- Values "increase" (allowing for rollover, i.e., modulo the field size except zero) according to a typical 'tick' time.
- A request is defined as TSval being non-zero, and a reply is defined as TSecr being non-zero.

- A receiver always responds to a request with the highest TSval received (allowing for rollover), which is not necessarily the most recently received.

Rollover can be handled as a special case or more completely using sequence number extension [RFC9187]; however, zero values need to be avoided explicitly.

>> TIME values **MUST NOT** use zeros as valid time values, because they are used as indicators of requests and responses.

TIME is reported to the user, whether used per-datagram or per- fragment (as defined in Section 11.4). When used per-fragment, the reported value is the minimum and maximum of each of the timestamp values received per-fragment.

>> Use of TIME per-fragment is **NOT RECOMMENDED**. Exceptions include supporting diagnostics on the reassembly process itself, which could be more appropriate to handle within the UDP option processing implementation.

11.9. Authentication (AUTH), RESERVED Only

The Authentication (AUTH, Kind=9) option is reserved for all UDP authentication mechanisms [To24]. AUTH is expected to cover the UDP user data and UDP options, with possible additional coverage of the IP pseudoheader and UDP header and potentially also support for NAT traversal (i.e., by zeroing the remote socket -- the source IP address and UDP port -- before computing the check), the latter in a similar manner as per TCP Authentication Option (TCP-AO) NAT traversal [RFC6978].

Like APC, AUTH is a SAFE option because it does not modify the UDP user data. AUTH could fail even where the user data has not been corrupted, such as when its contents have been overwritten. Such overwrites could be intentional and not widely known; defaulting to silent ignore ensures that option-aware endpoints do not change how users or applications operate unless explicitly directed to do otherwise. When a socket pair relies on AUTH, e.g., upon configuration of a security policy, this default is expected to be overridden, where incoming packets without AUTH or with a failed AUTH check would be silently dropped, such that only authenticated packets would be sent to the user. This approach enables security checks for AUTH to occur above UDP, in a separate shim layer or application library.

A specification for using AUTH is expected to define the coordination of AUTH security parameters and configuration of the socket pair when those parameters are installed. That specification is expected to address rules for when AUTH is required upon transmission and when the presence and correct validation of AUTH is required on reception.

11.10. Experimental (EXP)

The Experimental (EXP, Kind=127) option is allocated for experiments [RFC3692]. Only one such value is allocated because experiments are expected to use an Experimental ID (ExID) to differentiate concurrent use for different purposes, using UDP ExIDs registered with IANA according to the approach developed for TCP experimental options [RFC6994].

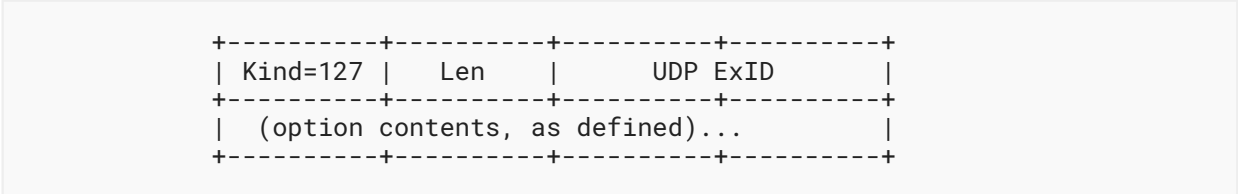

Figure 17: UDP EXP Option Format

>> The length of the Experimental option **MUST** be at least 4 to account for the Kind, Len, and 16-bit UDP ExID (similar to TCP ExIDs [RFC6994]).

The UDP EXP option uses only 16-bit ExIDs, unlike TCP ExIDs. In TCP, the first 16 bits of the ExID is unique; the additional 16 bits, where present, are used to decrease the chance of the entire ExID occurring in legacy use of the TCP EXP option. This extended variant provides no similar use for UDP EXP because ExIDs are required.

The UDP EXP option also includes an extended length format, where the option Len is 255, followed by two bytes of extended length.

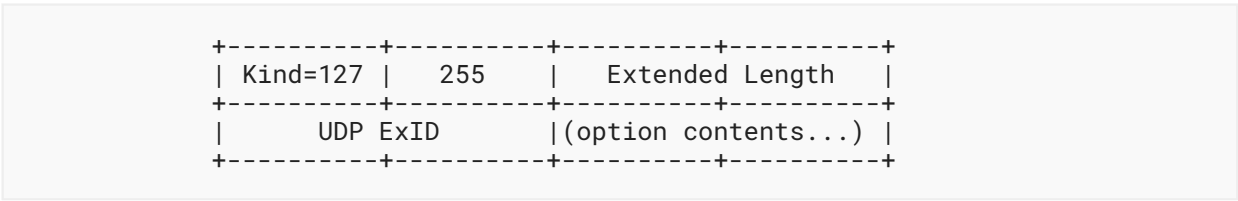

Figure 18: UDP EXP Extended Option Format

Assigned UDP Experimental IDs (ExIDs) are assigned from a combined TCP/UDP ExID registry managed by IANA (see Section 26). Assigned ExIDs can be used in either the EXP or UEXP options (see Section 12.3 for the latter).

12. UNSAFE Options

UNSAFE options are not safe to ignore and can be used unidirectionally or without soft-state confirmation of UDP option capability. They are always used only when the user data occurs inside a reassembled set of one or more UDP fragments, such that if UDP fragmentation is not supported, the enclosed UDP user data would be silently dropped anyway.

>> Applications using UNSAFE options **SHOULD NOT** also use zero-length UDP packets as signals, because they will arrive when UNSAFE options fail. Those that choose to allow such packets **MUST** account for such events.

>> UNSAFE options **MUST** be used only as part of UDP fragments, used either per-fragment or after reassembly.

>> Receivers supporting UDP options **MUST** silently drop the UDP user data of the reassembled datagram if any fragment or the entire datagram includes an UNSAFE option whose UKind is not supported or if an UNSAFE option appears outside the context of a fragment or reassembled fragments.

12.1. UNSAFE Compression (UCMP)

The UNSAFE Compression (UCMP, Kind=192) option is reserved for all UDP compression mechanisms. UCMP is expected to cover the UDP user data and some (e.g., later or in sequence) UDP options.

12.2. UNSAFE Encryption (UENC)

The UNSAFE Encryption (UENC, Kind=193) option is reserved for all UDP encryption mechanisms. UENC is expected to provide all of the services of the AUTH option (Section 11.9) and in addition to encrypt the UDP user data and some (e.g., later or in sequence) UDP options, in a similar manner as TCP Authentication Option Encryption (TCP-AO-ENC) [To18].

12.3. UNSAFE Experimental (UEXP)

The UNSAFE Experimental (UEXP, Kind=254) option is reserved for experiments [RFC3692]. As with EXP, only one such UEXP value is reserved because experiments are expected to use an Experimental ID (ExIDs) to differentiate concurrent use for different purposes, using UDP ExIDs registered with IANA according to the approach developed for TCP experimental options [RFC6994].

Assigned ExIDs can be used with either the UEXP or EXP options.

13. Rules for Designing New Options

The UDP option Kind space allows for the definition of new options; however, the currently defined options (including AUTH, UENC, and UCMP) do not allow for arbitrary new options. The following is a summary of rules for new options and their rationales:

>> New options **MUST NOT** be defined as "must-implement", i.e., they are not eligible for the asterisk ("*") designation used in Section 10.

This document defines the minimum set of "must-implement" UDP options. All new options are included at the discretion of a given implementation.

>> New options **MUST NOT** modify the content of options that precede them (in order of appearance and thus processing).

>> The fields of new options **MUST NOT** depend on the content of other options.

UNSAFE options can both depend on and vary user data content because they are contained only inside UDP fragments and thus are processed only by receivers capable of handling UDP options.

>> New options **MUST NOT** declare their order relative to other options, whether new or old, even as a preference.

>> At the sender, new options **MUST NOT** modify UDP packet content anywhere except within their option field, except only those contained within the UNSAFE option; areas that need to remain unmodified include the IP header, IP options, UDP user data, and surplus area (i.e., other options).

>> Options **MUST NOT** be modified in transit. This includes those already defined as well as new options.

>> New options **MUST NOT** require or allow that any UDP options (including themselves) or the remaining surplus area be modified in transit.

>> All options **MUST** indicate whether they can be used per-fragment and, if so, **MUST** also indicate how their success or failure is reported to the user. This document RECOMMENDS that options be useful per-fragment and also RECOMMENDS that options used per- fragment be reported to the user as a finite aggregate (e.g., a sum, a flag, etc.) rather than individually.

Note that only certain of the initially defined options violate these rules:

- >> The FRAG option modifies UDP user data, splitting it across multiple IP packets. UNSAFE options **MAY** modify the UDP user data, e.g., by encryption, compression, or other transformations. All other (SAFE) options **MUST NOT** modify the UDP user data.

14. Option Inclusion and Processing

The following rules apply to option inclusion by senders and processing by receivers.

>> Senders **MAY** add any option, as configured by the API.

>> All "must-support" options **MUST** be processed by receivers, if present (presuming UDP options are supported at that receiver).

>> Non-"must-support" options **MAY** be ignored by receivers, if present, e.g., based on API settings.

>> All options **MUST** be processed by receivers in the order encountered in the options area.

>> Unless configuration settings direct otherwise, all options except UNSAFE options **MUST** result in the UDP user data being passed to the upper layer protocol or application, regardless of whether all options are processed, are supported, or succeed.

The basic premise is that, for options-aware endpoints, the sender decides what options to add and the receiver decides what options to handle. Simply adding an option does not force work upon a receiver, with the exception of the "must-support" options.

Upon receipt, the receiver checks various properties of the UDP packet and its options to decide whether to accept or drop the UDP packet and whether to accept or ignore some of its options as follows (in order):

```
if the UDP checksum fails then
    silently drop the entire UDP packet (per RFC 1122)
if the UDP checksum passes or is zero then
    if (OCS != 0 and OCS fails) or
       (OCS == 0 and UDP CS != 0) then
        deliver the UDP user data but ignore other options
        (this is required to emulate legacy behavior)
    if (OCS != 0 and OCS passes) or
       (OCS == 0 and UDP CS == 0) then
        deliver the UDP user data after parsing
        and processing the rest of the options,
        regardless of whether each is supported or succeeds
        (again, this is required to emulate legacy behavior)
```

The design of the UNSAFE options ensures that the resulting UDP data will be silently dropped in both legacy receivers and options-aware receivers that do not recognize those options. Again, note that this still results in the delivery of a zero-length UDP packet.

Options-aware receivers can drop UDP packets with option processing errors via either an override of the default UDP processing or at the application layer.

That is, all options are treated the same, in that the transmitter can add it as desired and the receiver has the option to require it or not. Only if it is required (e.g., by API configuration) would the receiver require it being present and correct.

That is, for all options:

- if the option is not required by the receiver, then UDP packets missing the option are accepted.
- if the option is required (e.g., by override of the default behavior at the receiver) and missing or incorrectly formed, silently drop the UDP packet.
- if the UDP packet is accepted (either because the option is not required or because it was required and correct), then pass the option with the UDP packet via the API. Note that FRAG, NOP, and EOL are not passed to the user (see Section 15).

>> Any options whose length exceeds that of the UDP packet (i.e., intending to use data that would have been beyond the surplus area) **SHOULD** be silently ignored (again to model legacy behavior).

15. UDP API Extensions

UDP currently specifies an Application Programming Interface (API), summarized as follows (with Unix-style command as an example) [RFC0768]:

- Method to create new receive ports
  - e.g., bind(handle, recvaddr(optional), recvport)
- Receive, which returns data octets, source port, and source address
  - e.g., recvfrom(handle, srcaddr, srcport, data)
- Send, which specifies data, source and destination addresses, and source and destination ports
  - e.g., sendto(handle, destaddr, destport, data)

This API is extended to support options as follows:

- Extend the method to create receive ports to include per-packet and per-fragment receive options that are required or omitted as indicated by the application.

  >> Datagrams not containing these required options **MUST** be silently dropped and **SHOULD** be logged.
- Extend the method to create receive ports to have a means to indicate that all packets containing UDP options that are received on a particular socket pair are to be discarded.

  >> The default value for the setting to drop all packets containing UDP options **MUST** be to process packets containing UDP options normally (i.e., not to discard them).
- Extend the receive function to indicate the per-packet options and their parameters as received with the corresponding received datagram. Note that per-fragment options are handled within the processing of each fragment.

  >> Options and their processing status (success/fail) **MUST** be available to the user (i.e., application layer or upper layer protocol/service), both for the packet and for the fragment set, except for FRAG, NOP, and EOL; those three options are handled within UDP option processing only. As a reminder (from Section 14), all options except UNSAFE options **MUST** result in the UDP user data being passed to the application layer (unless overridden in the API), regardless of whether all options are processed, supported, or succeed.
- For fragments, success for an option is reported only when all fragments succeed for that option.

  >> Per-fragment option status reporting **SHOULD** default as needed (e.g., not computed and/or not passed up to the upper layers) to minimize overhead unless actively requested (e.g., by the user/application layer).

>> SAFE options associated with fragments are accumulated when associated with the reassembled packet; values **MAY** be coalesced, e.g., to indicate that only an AUTH failure of a fragment occurred, rather than not indicating the AUTH status of each fragment.

- Extend the send function to indicate the options to be added to the corresponding sent datagram. This includes indicating which options apply to individual fragments vs. which apply to the UDP packet prior to fragmentation, if fragmentation is enabled. This includes a minimum datagram length, such that the options list ends in EOL and additional space is zero-filled as needed. It also includes a maximum fragment size, e.g., as discovered by DPLPMTUD, whether implemented at the application layer per [RFC8899] or in conjunction with other UDP options [RFC9869].

Examples of API instances for Linux and FreeBSD are provided in Appendix A to encourage uniform cross-platform implementations.

APIs are not intended to provide user control over option order, especially on a per-packet basis, as this could create a covert channel (see Section 25). Similarly, APIs are not intended to provide user/application control over UDP fragment boundaries on a per-packet basis; although, they are expected to allow control over which options, including fragmentation, are enabled (or disabled) on a per-packet basis. Such control over fragmentation is critical to DPLPMTUD.

16. UDP Options Are for Transport, Not Transit

UDP options are indicated in the surplus area of the IP payload that is not used by UDP. That area is really part of the IP payload, not the UDP payload, and as such, it might be tempting to consider whether this is a generally useful approach to extending IP.

Unfortunately, the surplus area exists only for transports that include their own transport layer payload length indicator. TCP and SCTP include header length fields that already provide space for transport options by indicating the total length of the header area, such that the entire remaining area indicated in the network layer (IP) is the transport payload. UDP-Lite already uses the UDP Length field to indicate the boundary between data covered by the transport checksum and data not covered, and so there is no remaining area where the length of the UDP-Lite payload as a whole can be indicated [RFC3828].

UDP options are transport options. They are no more (or less) appropriate to be modified in-transit than any other portion of the transport datagram.

>> Generally, transport headers, options, and data are not intended to be modified in-transit. UDP options are no exception and are specified here as "**MUST NOT** be altered in transit".

However, note that the UDP option mechanism provides no specific protection against in-transit modification of the UDP header, UDP payload, or surplus area, except as provided by the OCS or the options selected (e.g., AUTH or UENC).

Unless protected by encryption (e.g., UENC or via other layers, like IPsec), UDP options remain visible to devices on the network path. The decision to not require mandatory encryption for UDP options to prevent such visibility was made because the key distribution and management

infrastructure necessary to support such encryption does not exist in many of the deployment scenarios of interest, notably those that use UDP directly as a stateless and connectionless transport protocol (e.g., see [He24]).

17. UDP Options vs. UDP-Lite

UDP-Lite provides partial checksum coverage so that UDP packets with errors in some locations can be delivered to the user [RFC3828]. It uses a different transport protocol number (136) than UDP (17) to interpret the UDP Length field as the prefix covered by the UDP checksum.

UDP (protocol 17) already defines the UDP Length field as the limit of the UDP checksum but by default also limits the data provided to the application as that which precedes the UDP Length. A goal of UDP-Lite is to deliver data beyond UDP Length as a default, which is why a separate transport protocol number was required.

UDP options do not use or need a separate transport protocol number because the data beyond the UDP Length offset (surplus data) is not provided to the application by default. That data is interpreted exclusively within the UDP transport layer.

UDP-Lite cannot support UDP options, either as proposed here or in any other form, because the entire payload of the UDP packet is already defined as user data and there is no additional field in which to indicate a surplus area for options. The UDP Length field in UDP-Lite is already used to indicate the boundary between user data covered by the checksum and user data not covered.

18. Interactions with Legacy Devices

It has always been permissible for the UDP Length to be inconsistent with the IP transport payload length [RFC0768]. Such inconsistency has been utilized in UDP-Lite using a different transport number. There are no known systems that use this inconsistency for UDP [RFC3828]. It is possible that such use might interact with UDP options, i.e., where legacy systems might generate UDP datagrams that appear to have UDP options. The OCS provides protection against such events and is stronger than a static "magic number".

UDP options have been tested as interoperable with Linux, macOS, and Windows Cygwin and worked through NAT devices. These systems successfully delivered only the user data indicated by the UDP Length field and silently discarded the surplus area.

One reported embedded device passes the entire IP datagram to the UDP application layer. Although this feature could enable application-layer UDP option processing, it would require that conventional UDP user applications examine only the UDP user data.

This feature is also inconsistent with the UDP application interface [RFC0768] [RFC1122].

It has been reported that Alcatel-Lucent's "Brick" Intrusion Detection System has a default configuration that interprets inconsistencies between UDP Length and IP Length as an attack to be reported. Note that other firewall systems, e.g., Check Point, use a default "relaxed UDP length verification" to avoid falsely interpreting this inconsistency as an attack.

There are known uses of UDP exchanges of zero-length UDP user data packets, notably in the TIME protocol [RFC0868]. The need to support such packets is also noted in the UDP usage guidelines [RFC8085]. Some of the mechanisms in this document can generate more zero- length UDP packets for a UDP option-aware endpoint than for a legacy (non-aware) endpoint (e.g., based on some error conditions), and some can generate fewer (e.g., fragment reassembly). Because such packets inherently carry no unique transport header or transport content, endpoints are already expected to be tolerant of their (inadvertent) replication or loss by the network, so such variations are not expected to be problematic.

19. Options in a Stateless, Unreliable Transport Protocol

There are two ways to interpret options for a stateless, unreliable protocol -- an option is either local to the message or intended to affect a stream of messages in a soft-state manner. Either interpretation is valid for defined UDP options.

It is impossible to know in advance whether an endpoint supports a UDP option.

>> All UDP options other than UNSAFE ones **MUST** be ignored if not supported or upon failure (e.g., APC).

>> All UDP options that fail **MUST** result in the UDP data still being sent to the application layer by default to ensure equivalence with legacy devices.

UDP options that rely on soft-state exchange need to allow message reordering and loss, in the same way as UDP applications [RFC8085].

The above requirements prevent using any option that cannot be safely ignored unless it is hidden inside the FRAG area (i.e., UNSAFE options). Legacy systems also always need to be able to interpret the transport fragments as individual UDP packets.

20. UDP Option State Caching

Some TCP connection parameters, stored in the TCP Control Block (TCB), can be usefully shared either among concurrent connections or between connections in sequence, known as TCP Sharing [RFC9040]. Although UDP is stateless, some of the options proposed herein could have similar benefits in being shared or cached. We call this UCB sharing, or UDP Control Block sharing, by analogy. Just as TCB sharing is not a standard because it is consistent with existing TCP specifications, UCB sharing would be consistent with existing UDP specifications, including this one. Both are implementation issues that are outside the scope of their respective specifications, and so UCB sharing is outside the scope of this document.

21. Updates to RFC 768

This document updates [RFC0768] as follows:

- This document defines the meaning of the IP payload area beyond the UDP length but within the IP Length as the surplus area used herein for UDP options.
- This document extends the UDP API to support the use of UDP options.

22. Interactions with Other RFCs (and drafts)

This document clarifies the interaction between UDP Length and IP Length that is not explicitly constrained in either UDP or the host requirements [RFC0768] [RFC1122].

Teredo extensions (TEs) define use of a similar difference between these lengths for trailers [RFC4380] [RFC6081]. In [RFC6081], TE defines the length of an IPv6 payload inside UDP as pointing to less than the end of the UDP payload, enabling trailing options for that IPv6 packet:

> ...the IPv6 packet length (i.e., the Payload Length value in the IPv6 header plus the IPv6 header size) is less than or equal to the UDP payload length (i.e., the Length value in the UDP header minus the UDP header size)

UDP options are not affected by the difference between the UDP user payload end and the payload IPv6 end; both would end at the UDP user payload, which could end before the enclosing IPv4 or IPv6 header indicates -- allowing UDP options in addition to the trailer options of the IPv6 payload. The result, if UDP options were used, is shown in Figure 19.

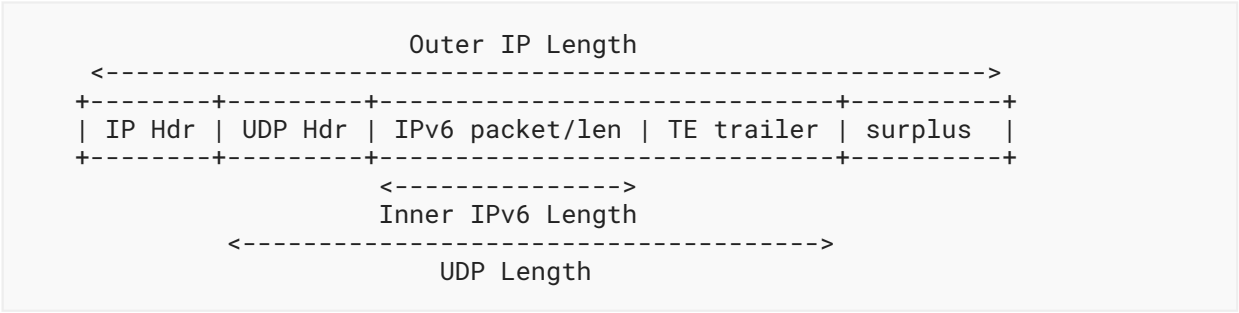

Figure 19: TE Trailers and UDP Options Used Concurrently

UDP options cannot be supported when a UDP packet has no independent UDP Length. One such case is when UDP Length==0 in IPv6, intended for (but not limited to) IPv6 Jumbograms [RFC2675]. Note that although this technique is "Standard", the specification did not "update" UDP [RFC0768]. Another such case arises when UDP is proxied via HTTP [RFC9298], as the UDP header is omitted and only the UDP user data is transported.

This document is consistent with the UDP profile for RObust Header Compression (ROHC) [RFC3095], noted here:

> The Length field of the UDP header **MUST** match the Length field(s) of the preceding subheaders, i.e., there must not be any padding after the UDP payload that is covered by the IP Length.

ROHC compresses UDP headers only when this match succeeds. It does not prohibit UDP headers where the match fails; in those cases, ROHC default rules (Section 5.10 of [RFC3095]) would cause the UDP header to remain uncompressed. Upon receipt of a compressed UDP header, Appendix A.1.3 of [RFC3095] indicates that the UDP length is "INFERRED"; in uncompressed packets, it would simply be explicitly provided.

This issue of handling UDP header compression is more explicitly described in more recent specifications, e.g., Section 10.10 of [RFC8724].

23. Multicast and Broadcast Considerations

UDP options are primarily intended for unicast use. Using these options over multicast or broadcast IP requires careful consideration, e.g., to ensure that the options used are safe for different endpoints to interpret differently (e.g., either to support or silently ignore) or to ensure that all receivers of a multicast or broadcast group confirm support for the options in use.

24. Network Management Considerations

UDP options use and configuration may be useful to track and manage remotely. IP Flow Information Export (IPFIX) [RFC7011] Information Elements for UDP options have been defined in [Bo24]. Similar to what has been done for TCP [RFC9648], a YANG model [RFC7950] for use by network management protocols (e.g., NETCONF [RFC6241] or RESTCONF [RFC8040]) may be developed. Development of these models is outside the scope of this document.

25. Security Considerations

There are a number of security issues raised by the introduction of options to UDP. Some are specific to this variant, but others are associated with any packet processing mechanism; all are discussed further in this section.

25.1. General Considerations Regarding the Use of Options

Note that any user application that considers UDP options to adversely affect security need not enable them. However, their use does not impact security in a substantially different way than TCP options; both enable the use of a control channel that has the potential for abuse. Similar to TCP, there are many options that, if unprotected, could be used by an attacker to interfere with communication.

UDP options are not covered by DTLS [RFC9147]. Neither TLS [RFC8446] (Transport Layer Security for TCP) nor DTLS (TLS for UDP) protect the transport layer; both operate as a shim layer solely on the user data of transport packets, protecting only their contents.

Just as TLS does not protect the TCP header or its options, DTLS does not protect the UDP header or the new options introduced by this document. Transport security is provided in TCP by the TCP Authentication Option (TCP-AO) [RFC5925] and (when defined) in UDP by the Authentication (AUTH) option (Section 11.9) and (when defined) the UNSAFE Encryption (UENC) option (Section 12). Transport headers are also protected as payload when using IP security (IPsec) [RFC4301].

Some UDP options are never passed to the receiving application, notably FRAG, NOP, and EOL. They are not intended to convey information, either by their presence (FRAG, EOL) or number (NOP). It could also be useful to provide the options received in a reference order (e.g., sorted by option number) to avoid the order of options being used as a covert channel.

All logging is rate limited to avoid logging itself becoming a resource vulnerability.

25.2. Considerations Regarding On-Path Attacks

UDP options, like any options, have the potential to expose option information to on-path attackers, unless the options themselves are encrypted (as might be the case with some configurations of UENC, when defined). Application protocol designers are expected to ensure that information in UDP options is not used with the assumption of privacy unless UENC provides that capability. Application protocol designers using secure payload contents (e.g., via DTLS) are expected to be aware that UDP options add information that is not inside the UDP payload and thus not protected by the same mechanism and that alternate mechanisms (again, as might be the case with some configurations of UENC) could be additionally required to protect against information disclosure.

>> Implementations concerned with the potential use of UDP options as a covert channel **MAY** consider limiting use of some or all options. Such implementations **SHOULD** return options in an order not related to their sequence in the received packet.

UDP options create new potential opportunities for Distributed DoS (DDos) attacks, notably through the use of fragmentation. When enabled, UDP options cause additional work at the receiver; however, of the "must-support" options, only REQ (e.g., when used with DPLPMTUD [RFC9869]) will cause the upper layer to initiate a UDP response in the absence of user transmission.

>> Implementations concerned with the potential for DoS attacks involving large numbers of UDP options, either implemented or unknown, or excessive sequences of valid repeating options (e.g., NOPs) **SHOULD** detect excessive numbers of such occurrences and limit resources they use, e.g., through silent packet drops. Such responses **SHOULD** be logged. Specific thresholds for such limits will vary based on implementation and are thus not included here.

25.3. Considerations Regarding Option Processing

UDP options use the TLV syntax similar to that of TCP. This syntax is known to require serial processing and could pose a DoS risk, e.g., if an attacker adds large numbers of unknown options that need to be parsed in their entirety, as is the case for IPv6 [RFC8504].

The use of UDP packets with inconsistent IP and UDP Length fields has the potential to trigger a buffer overflow error if not properly handled, e.g., if space is allocated based on the smaller field and copying is based on the larger field. However, there have been no reports of such vulnerability, and it would rely on inconsistent use of the two fields for memory allocation and copying.

Because required options come first and at most once each (with the exception of NOPs, which never need to come in sequences of more than seven in a row), their DoS impact is limited. Note that TLV formats for options do require serial processing, but any format that allows future options, whether ignored or not, could introduce a similar DoS vulnerability.

>> Implementations concerned with the potential for UDP options introducing a vulnerability **MAY** implement only the required UDP options and **SHOULD** also limit processing of TLVs, in number of non-padding options, total length, or both. The number of non-zero TLVs allowed in such cases **MUST** be at least as many as the number of concurrent options supported with an additional few to account for unexpected unknown options but **SHOULD** also consider being adaptive and based on the implementation to avoid locking in that limit globally.

For example, if a system supports 10 different option types that could concurrently be used, it is expected to allow up to around 13-14 different options in the same packet. This document avoids specifying a fixed minimum but recognizes that a given system might not expect to receive more than a few unknown option types per packet.

25.4. Considerations for Fragmentation

UDP fragmentation introduces its own set of security concerns, which can be handled in a manner similar to IP reassembly or TCP segment reordering [CERT18]. In particular, the number of UDP packets pending reassembly and effort used for reassembly is typically limited. In addition, it could be useful to assume a reasonable minimum fragment size, e.g., that non-terminal fragments are never be smaller than 500 bytes.

>> Implementations concerned with the potential for UDP fragmentation introducing a vulnerability **SHOULD** implement limits on the number of pending fragments.

25.5. Considerations for Providing UDP Security

UDP security is not intended to rely solely on transport layer processing of options. UNSAFE options are the only type that share fate with the UDP data because of the way that data is hidden in the surplus area until after those options are processed. All other options default to being

silently ignored at the transport layer but could be dropped if that default is either overridden (e.g., by configuration) or discarded at the application layer (e.g., using information about the options processed that are passed along with the UDP packet).

Options providing UDP security, e.g., AUTH and UENC, require endpoint key and security parameter coordination, which UDP options (being stateless) do not facilitate. These parameters include whether and when to override the defaults described herein, especially at the transmitter as to when emitted packets need to include AUTH and at the receiver as to whether (and when) packets with failed AUTH and/or without AUTH (or that fail the AUTH checks) are not to be forwarded to the user/application.

25.6. Considerations Regarding Middleboxes

Some middleboxes operate as UDP relays, forwarding data between a UDP socket and another transport socket by modifying the IP and/or UDP headers without properly acting as a protocol endpoint (i.e., an application layer proxy). In such cases, a sender might add UDP options that could be stripped by the middlebox before the packet is forwarded to the second socket. A remote application will not receive the options (for SAFE options, the payload data will be received; for UNSAFE options, the payload data will not be received). In such cases, the application will function as it would if communicating with a remote endpoint that does not support UDP options.

Additionally, [Zu20] reports that packets containing UDP options do not traverse certain Internet paths; most likely, those options were stripped (e.g., by resetting the IP Length to correspond to the UDP length, truncating the surplus area) or packets with options were dropped. UDP options do not function over such paths.

26. IANA Considerations

IANA has created the "User Datagram Protocol (UDP)" registry group, which consists of the "UDP Option Kind Numbers" registry and a pointer to the unified "TCP/UDP Experimental Option Experiment Identifiers (TCP/UDP ExIDs)" registry. Note that the "TCP experimental IDs (ExIDs)" registry has been renamed as the "TCP/UDP Experimental Option Experiment Identifiers (TCP/UDP ExIDs)" registry, and is a unified registry for both TCP and UDP ExIDs. IANA has added the following note to the unified TCP/UDP ExID registry:

> Note 16-bit ExIDs can be used with either TCP or UDP; 32-bit ExIDs can be used with TCP or their first 16 bits can be used with UDP. Use with each transport (TCP, UDP) is indicated in the protocol column, as defined in RFC 9868.

Initial values of the UDP Option Kind registry are as listed in Section 10, including those both assigned and reserved. Additional values in this registry are to be assigned from the Unassigned values in Section 10 by IESG Approval or Standards Action [RFC8126]. Those assignments are subject to the conditions set forth in this document, particularly (but not limited to) those in Section 13.

>> Although option nicknames are not used in-band, new UNSAFE option names **MUST** commence with the capital letter "U" and new SAFE options **MUST NOT** commence with either uppercase or lowercase "U".

IANA has added the following note to the "UDP Option Kind Numbers" indicating entries are mandatory to implement when UDP options are supported. No new options may be created that are mandatory to implement in all UDP options implementations.

> Codepoints 0-7 **MUST** be supported on any implementation supporting UDP options. All others are supported at the discretion of each implementation.

UDP Experimental Option Experiment Identifiers (UDP ExIDs) are intended for use in a similar manner as TCP ExIDs [RFC6994]. Both TCP and UDP ExIDs are managed as a single, unified registry because such options could be used for both transport protocols and because the option space is large enough that there is no clear need to maintain them separately. This new TCP/UDP ExIDs registry has entries for both transports, although each codepoint needs to be explicitly defined for each transport protocol in which it is used, i.e., defining a codepoint in TCP does not imply it has a similar use in UDP. IANA has added a "Protocol" field to the registry and updated the current TCP ExIDs to be indicated as defined for TCP. New assignments are to indicate the transport for which it is defined.

TCP/UDP ExIDs can be used in either (or both) the UDP EXP (Section 11.10) or UEXP (Section 12.3) options. TCP/UDP ExID entries for use in UDP consist of a 16-bit ExID (in network-standard order), and (as with the original TCP ExIDs) will preferentially also include a short description and acronym for use in documentation. TCP/UDP ExIDs used for UDP are always 16 bits because their use in EXP and UEXP options is required and thus do not need a larger codepoint value to decrease the probability of accidental occurrence with non-ExID uses of the experimental options, as is the case with TCP ExIDs (e.g., when using 32-bit ExIDs). ExIDs defined solely for TCP options could be either 16 or 32 bits and all ExIDs (including now UDP) need to be unique in their first 16 bits, as originally described for TCP [RFC6994].

Values in the TCP/UDP ExID registry are to be assigned by IANA using first-come, first-served (FCFS) rules applied to both the ExID value and the acronym [RFC8126]. UDP options using these ExIDs are subject to the same conditions as new UDP options, i.e., they too are subject to the conditions set forth in this document, particularly (but not limited to) those in Section 13.

27. References

27.1. 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>.

[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>.

[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>.

[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>.

[RFC9869] Fairhurst, G. and T. Jones, "Datagram PLPMTUD for UDP Options", RFC 9869, DOI 10.17487/RFC9869, September 2025, <https://www.rfc-editor.org/info/rfc9869>.

27.2. Informative References

[Bo24] Boucadair, M. and T. Reddy.K, "Export of UDP Options Information in IP Flow Information Export (IPFIX)", Work in Progress, Internet-Draft, draft-ietf-opsawg-tsvwg-udp-ipfix-14, 22 July 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-opsawg-tsvwg-udp-ipfix-14>.

[CERT18] CERT Coordination Center, "TCP implementations vulnerable to Denial of Service", Vulnerability Note VU#962459, Software Engineering Institute, CMU, 2018, <https://www.kb.cert.org/vuls/id/962459>.

[Fa18] Fairhurst, G., Jones, T., and R. Zullo, "Checksum Compensation Options for UDP Options", Work in Progress, Internet-Draft, draft-fairhurst-udp-options-cco-00, 19 October 2018, <https://datatracker.ietf.org/doc/html/draft-fairhurst-udp-options-cco-00>.

[He24] Heard, C. M., "Use of UDP Options for Transmission of Large DNS Responses", Work in Progress, Internet-Draft, draft-heard-dnsop-udp-opt-large-dns-responses-00, 28 April 2024, <https://datatracker.ietf.org/doc/html/draft-heard-dnsop-udp-opt-large-dns-responses-00>.

[Hi15] Hildebrand, J. and B. Trammell, "Substrate Protocol for User Datagrams (SPUD) Prototype", Work in Progress, Internet-Draft, draft-hildebrand-spud-prototype-03, 9 March 2015, <https://datatracker.ietf.org/doc/html/draft-hildebrand-spud-prototype-03>.

[La78] Lamport, L., "Time, clocks, and the ordering of events in a distributed system", Communications of the ACM, vol. 21, no. 7, pp. 558-565, DOI 10.1145/359545.359563, July 1978, <https://doi.org/10.1145/359545.359563>.

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

[RFC0868] Postel, J. and K. Harrenstien, "Time Protocol", STD 26, RFC 868, DOI 10.17487/RFC0868, May 1983, <https://www.rfc-editor.org/info/rfc868>.

[RFC1071] Braden, R., Borman, D., and C. Partridge, "Computing the Internet checksum", RFC 1071, DOI 10.17487/RFC1071, September 1988, <https://www.rfc-editor.org/info/rfc1071>.

[RFC1141] Mallory, T. and A. Kullberg, "Incremental updating of the Internet checksum", RFC 1141, DOI 10.17487/RFC1141, January 1990, <https://www.rfc-editor.org/info/rfc1141>.

[RFC1191] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, DOI 10.17487/RFC1191, November 1990, <https://www.rfc-editor.org/info/rfc1191>.

[RFC2675] Borman, D., Deering, S., and R. Hinden, "IPv6 Jumbograms", RFC 2675, DOI 10.17487/RFC2675, August 1999, <https://www.rfc-editor.org/info/rfc2675>.

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

[RFC3095] Bormann, C., Burmeister, C., Degermark, M., Fukushima, H., Hannu, H., Jonsson, L., Hakenberg, R., Koren, T., Le, K., Liu, Z., Martensson, A., Miyazaki, A., Svanbro, K., Wiebke, T., Yoshimura, T., and H. Zheng, "RObust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed", RFC 3095, DOI 10.17487/RFC3095, July 2001, <https://www.rfc-editor.org/info/rfc3095>.

[RFC3173] Shacham, A., Monsour, B., Pereira, R., and M. Thomas, "IP Payload Compression Protocol (IPComp)", RFC 3173, DOI 10.17487/RFC3173, September 2001, <https://www.rfc-editor.org/info/rfc3173>.

[RFC3385] Sheinwald, D., Satran, J., Thaler, P., and V. Cavanna, "Internet Protocol Small Computer System Interface (iSCSI) Cyclic Redundancy Check (CRC)/Checksum Considerations", RFC 3385, DOI 10.17487/RFC3385, September 2002, <https://www.rfc-editor.org/info/rfc3385>.

[RFC3692] Narten, T., "Assigning Experimental and Testing Numbers Considered Useful", BCP 82, RFC 3692, DOI 10.17487/RFC3692, January 2004, <https://www.rfc-editor.org/info/rfc3692>.

[RFC3828] Larzon, L., Degermark, M., Pink, S., Jonsson, L., Ed., and G. Fairhurst, Ed., "The Lightweight User Datagram Protocol (UDP-Lite)", RFC 3828, DOI 10.17487/RFC3828, July 2004, <https://www.rfc-editor.org/info/rfc3828>.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <https://www.rfc-editor.org/info/rfc4301>.

[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>.

[RFC4380] Huitema, C., "Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)", RFC 4380, DOI 10.17487/RFC4380, February 2006, <https://www.rfc-editor.org/info/rfc4380>.

[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, <https://www.rfc-editor.org/info/rfc4787>.

[RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <https://www.rfc-editor.org/info/rfc5925>.

[RFC6081] Thaler, D., "Teredo Extensions", RFC 6081, DOI 10.17487/RFC6081, January 2011, <https://www.rfc-editor.org/info/rfc6081>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

[RFC6864] Touch, J., "Updated Specification of the IPv4 ID Field", RFC 6864, DOI 10.17487/RFC6864, February 2013, <https://www.rfc-editor.org/info/rfc6864>.

[RFC6935] Eubanks, M., Chimento, P., and M. Westerlund, "IPv6 and UDP Checksums for Tunneled Packets", RFC 6935, DOI 10.17487/RFC6935, April 2013, <https://www.rfc-editor.org/info/rfc6935>.

[RFC6978] Touch, J., "A TCP Authentication Option Extension for NAT Traversal", RFC 6978, DOI 10.17487/RFC6978, July 2013, <https://www.rfc-editor.org/info/rfc6978>.

[RFC6994] Touch, J., "Shared Use of Experimental TCP Options", RFC 6994, DOI 10.17487/RFC6994, August 2013, <https://www.rfc-editor.org/info/rfc6994>.

[RFC7011] Claise, B., Ed., Trammell, B., Ed., and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information", STD 77, RFC 7011, DOI 10.17487/RFC7011, September 2013, <https://www.rfc-editor.org/info/rfc7011>.

[RFC7323] Borman, D., Braden, B., Jacobson, V., and R. Scheffenegger, Ed., "TCP Extensions for High Performance", RFC 7323, DOI 10.17487/RFC7323, September 2014, <https://www.rfc-editor.org/info/rfc7323>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[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>.

[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>.

[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>.

[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>.

[RFC8504] Chown, T., Loughney, J., and T. Winters, "IPv6 Node Requirements", BCP 220, RFC 8504, DOI 10.17487/RFC8504, January 2019, <https://www.rfc-editor.org/info/rfc8504>.

[RFC8724] Minaburo, A., Toutain, L., Gomez, C., Barthel, D., and JC. Zuniga, "SCHC: Generic Framework for Static Context Header Compression and Fragmentation", RFC 8724, DOI 10.17487/RFC8724, April 2020, <https://www.rfc-editor.org/info/rfc8724>.

[RFC8899] Fairhurst, G., Jones, T., Tüxen, M., Rüngeler, I., and T. Völker, "Packetization Layer Path MTU Discovery for Datagram Transports", RFC 8899, DOI 10.17487/RFC8899, September 2020, <https://www.rfc-editor.org/info/rfc8899>.

[RFC9040] Touch, J., Welzl, M., and S. Islam, "TCP Control Block Interdependence", RFC 9040, DOI 10.17487/RFC9040, July 2021, <https://www.rfc-editor.org/info/rfc9040>.

[RFC9147] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", RFC 9147, DOI 10.17487/RFC9147, April 2022, <https://www.rfc-editor.org/info/rfc9147>.

[RFC9187] Touch, J., "Sequence Number Extension for Windowed Protocols", RFC 9187, DOI 10.17487/RFC9187, January 2022, <https://www.rfc-editor.org/info/rfc9187>.

[RFC9260] Stewart, R., Tüxen, M., and K. Nielsen, "Stream Control Transmission Protocol", RFC 9260, DOI 10.17487/RFC9260, June 2022, <https://www.rfc-editor.org/info/rfc9260>.

[RFC9293] Eddy, W., Ed., "Transmission Control Protocol (TCP)", STD 7, RFC 9293, DOI 10.17487/RFC9293, August 2022, <https://www.rfc-editor.org/info/rfc9293>.

[RFC9298] Schinazi, D., "Proxying UDP in HTTP", RFC 9298, DOI 10.17487/RFC9298, August 2022, <https://www.rfc-editor.org/info/rfc9298>.

[RFC9648] Scharf, M., Jethanandani, M., and V. Murgai, "YANG Data Model for TCP", RFC 9648, DOI 10.17487/RFC9648, October 2024, <https://www.rfc-editor.org/info/rfc9648>.

[To18] Touch, J. D., "A TCP Authentication Option Extension for Payload Encryption", Work in Progress, Internet-Draft, draft-touch-tcp-ao-encrypt-09, 19 July 2018, <https://datatracker.ietf.org/doc/html/draft-touch-tcp-ao-encrypt-09>.

[To24] Touch, J. D., "The UDP Authentication Option", Work in Progress, Internet-Draft, draft-touch-tsvwg-udp-auth-opt-00, 3 March 2024, <https://datatracker.ietf.org/doc/html/draft-touch-tsvwg-udp-auth-opt-00>.

[Zu20] Zullo, R., Jones, T., and G. Fairhurst, "Overcoming the Sorrows of the Young UDP Options", 4th Network Traffic Measurement and Analysis Conference (TMA), 2020, <https://dl.ifip.org/db/conf/tma/tma2020/tma2020-camera-paper70.pdf>.

Appendix A. Implementation Information

The following information is provided to encourage consistent naming for API implementations.

System-level variables (sysctl):

Name	Default	Meaning
net.ipv4.udp_opt	0	UDP options available
net.ipv4.udp_opt_ocs	1	Use OCS
net.ipv4.udp_opt_apc	0	Include APC
net.ipv4.udp_opt_frag	0	Fragment
net.ipv4.udp_opt_mds	0	Include MDS
net.ipv4.udp_opt_mrds	0	Include MRDS
net.ipv4.udp_opt_req	0	Include REQ
net.ipv4.udp_opt_resp	0	Include RES
net.ipv4.udp_opt_time	0	Include TIME
net.ipv4.udp_opt_auth	0	Include AUTH
net.ipv4.udp_opt_exp	0	Include EXP
net.ipv4.udp_opt_ucmp	0	Include UCMP
net.ipv4.udp_opt_uenc	0	Include UENC

Name	Default	Meaning
net.ipv4.udp_opt_uexp	0	Include UEXP

Table 2

Socket options (sockopt), cached for outgoing datagrams:

Name	Meaning
UDP_OPT	Enable UDP options (at all)
UDP_OPT_OCS	Use UDP OCS
UDP_OPT_APC	Enable UDP APC option
UDP_OPT_FRAG	Enable UDP fragmentation
UDP OPT MDS	Enable UDP MDS option
UDP OPT MRDS	Enable UDP MRDS option
UDP OPT REQ	Enable UDP REQ option
UDP OPT RES	Enable UDP RES option
UDP_OPT_TIME	Enable UDP TIME option
UDP OPT AUTH	Enable UDP AUTH option
UDP OPT EXP	Enable UDP EXP option
UDP_OPT_UCMP	Enable UDP UCMP option
UDP_OPT_UENC	Enable UDP UENC option
UDP OPT UEXP	Enable UDP UEXP option

Table 3

Send/sendto parameters:

- (Same as sysctl, with different prefixes)

Connection parameters (per-socket pair cached state, part UCB):

Name	Initial Value
opts_enabled	net.ipv4.udp_opt

Name	Initial Value
ocs_enabled	net.ipv4.udp_opt_ocs

Table 4

NB: The JUNK option is included for debugging purposes and is not intended to be enabled otherwise.

System variables:

net.ipv4.udp_opt_junk 0

System-level variables (sysctl):

Name	Default	Meaning
net.ipv4.udp_opt_junk	0	Default use of junk

Table 5

Socket options (sockopt):

Name	Params	Meaning
UDP_JUNK	-	Enable UDP junk option
UDP_JUNK_VAL	fillval	Value to use as junk fill
UDP_JUNK_LEN	length	Length of junk payload in bytes

Table 6

Connection parameters (per-socket pair cached state, part UCB):

Name	Initial Value
junk_enabled	net.ipv4.udp_opt_junk
junk_value	0xABCD
junk_len	4

Table 7

Acknowledgments

This work benefitted from feedback from Erik Auerswald, Bob Briscoe, Ken Calvert, Ted Faber, Gorry Fairhurst (including OCS for errant middlebox traversal), C. M. Heard (editor of this document, including combining previous FRAG and LITE options into the new FRAG, as well as Figure 12), Tom Herbert, Tom Jones, Mark Smith, Carl Williams, and Raffaele Zullo, as well as discussions on the IETF TSVWG and SPUD email lists.

This work was partly supported by USC/ISI's Postel Center.

Authors' Addresses

Joe Touch
Independent Consultant
Manhattan Beach, CA 90266
United States of America
Phone: +1 (310) 560-0334
Email: touch@strayalpha.com

C. M. (Mike) Heard (EDITOR)
Unaffiliated
PO Box 2667
Redwood City, CA 94064-2667
United States of America
Phone: +1 (408) 499-7257
Email: heard@pobox.com