

TSVWG
Internet Draft
Intended status: Standards Track
Intended updates: 768
Expires: September 2022

J. Touch
Independent Consultant
March 26, 2022

Transport Options for UDP
draft-ietf-tsvwg-udp-options-18.txt

Abstract

Transport protocols are extended through the use of transport header options. This document extends UDP by indicating the location, syntax, and semantics for UDP transport layer options.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

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

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

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

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

This Internet-Draft will expire on September 26, 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the [Trust Legal Provisions](#) and are provided without warranty as described in the Revised BSD License.

Table of Contents

| | | |
|-----------------------|--|--------------------|
| 1. | Introduction..... | 3 |
| 2. | Conventions used in this document..... | 3 |
| 3. | Terminology..... | 3 |
| 4. | Background..... | 4 |
| 5. | The UDP Option Area..... | 5 |
| 6. | The UDP Surplus Area Structure..... | 8 |
| 7. | The Option Checksum (OCS)..... | 8 |
| 8. | UDP Options..... | 10 |
| 9. | Safe UDP Options..... | 13 |
| 9.1. | End of Options List (EOL)..... | 13 |
| 9.2. | No Operation (NOP)..... | 14 |
| 9.3. | Alternate Payload Checksum (APC)..... | 14 |
| 9.4. | Fragmentation (FRAG)..... | 16 |
| 9.5. | Maximum Datagram Size (MDS)..... | 19 |
| 9.6. | Maximum Reassembled Datagram Size (MRDS)..... | 20 |
| 9.7. | Echo request (REQ) and echo response (RES)..... | 21 |
| 9.8. | Timestamps (TIME)..... | 21 |
| 9.9. | Authentication (AUTH)..... | 22 |
| 9.10. | Experimental (EXP)..... | 23 |
| 10. | UNSAFE Options..... | 24 |
| 10.1. | UNSAFE Encryption (UENC)..... | 25 |
| 10.2. | UNSAFE Experimental (UEXP)..... | 25 |
| 11. | Rules for designing new options..... | 25 |
| 12. | Option inclusion and processing..... | 26 |
| 13. | UDP API Extensions..... | 28 |
| 14. | UDP Options are for Transport, Not Transit..... | 29 |
| 15. | UDP options vs. UDP-Lite..... | 29 |
| 16. | Interactions with Legacy Devices..... | 30 |
| 17. | Options in a Stateless, Unreliable Transport Protocol..... | 30 |
| 18. | UDP Option State Caching..... | 31 |
| 19. | Updates to RFC 768..... | 31 |
| 20. | Interactions with other RFCs (and drafts)..... | 32 |
| 21. | Multicast Considerations..... | 33 |

| | | |
|-----------------------|--|--------------------|
| 22. | Security Considerations..... | 33 |
| 23. | IANA Considerations..... | 34 |
| 24. | References..... | 35 |
| 24.1. | Normative References..... | 35 |

Touch

Expires September 26, 2022

[Page 2]

Internet-Draft

Transport Options for UDP

March 2022

| | | |
|-----------------------------|---|--------------------|
| 24.2. | Informative References..... | 35 |
| 25. | Acknowledgments..... | 38 |
| Appendix A. | Implementation Information..... | 39 |

[1.](#) Introduction

Transport protocols use options as a way to extend their capabilities. TCP [[RFC793](#)], SCTP [[RFC4960](#)], and DCCP [[RFC4340](#)] include space for these options but UDP [[RFC768](#)] currently does not. This document defines 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.

[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) indicates 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:

- o IP datagram [[RFC791](#)][[RFC8200](#)] - an IP packet, composed of the IP header and an IP payload area
- o 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
- o UDP packet - the more contemporary term used herein to refer to a

user datagram [[RFC768](#)]

- o Surplus area - the area of an IP payload that follows a UDP packet; this area is used for UDP options in this document

Touch

Expires September 26, 2022

[Page 3]

Internet-Draft

Transport Options for UDP

March 2022

- o UDP fragment - one or more components of a UDP packet and its UDP options that enables transmission as IP payloads larger than permitted by IP datagram maximum sizes; note that each UDP fragment is itself transmitted as a UDP packet with its own options
- o (UDP) User data - the user data field of a UDP packet [[RFC768](#)]
- o UDP Length - the length field of a UDP header [[RFC768](#)]
- o Must-support options - UDP options that all implementations are required to support. Their use in individual UDP packets is optional.

[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, Window Scale, Timestamp, and Authentication Options [[RFC793](#)] [[RFC5925](#)] [[RFC7323](#)].

Header options are used both in stateful (connection-oriented, e.g., TCP [[RFC793](#)], SCTP [[RFC4960](#)], DCCP [[RFC4340](#)]) and stateless (connectionless, e.g., IPv4 [[RFC791](#)], 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 [[RFC768](#)]. The UDP header was intended to be a minimal

addition to IP, providing only ports 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, SCTP). There are a number of reasons why Internet historians suggest that 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 concurrent design of multi-segment multiplexing protocols, however, so 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 [\[RFC6081\]](#) uses values of the UDP Length that are larger than the IP payload as an additional type of signal, as noted in [Section 20](#). UDP options uses a value smaller than the IP payload to enable backwards compatibility with existing UDP implementations, i.e., to deliver the UDP Length of UDP user data to the application and silently ignore the additional surplus area data. Using a value larger than the IP payload could either be considered malformed (and ought to be silently dropped by UDP processing) or could cause buffer overruns, and so is not considered silently and safely backward compatible.

[5](#). 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 detail in [Section 16](#)). The UDP Option area is created when the UDP Length indicates a smaller transport payload than implied by the IP header.

For IPv4, IP Total Length field indicates the total IP datagram length (including 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), as shown in Figure 1 [[RFC791](#)]. As a result, the typical (and largest valid) value for UDP Length is:

$$\text{UDP_Length} = \text{IPv4_Total_Length} - \text{IPv4_IHL} * 4$$

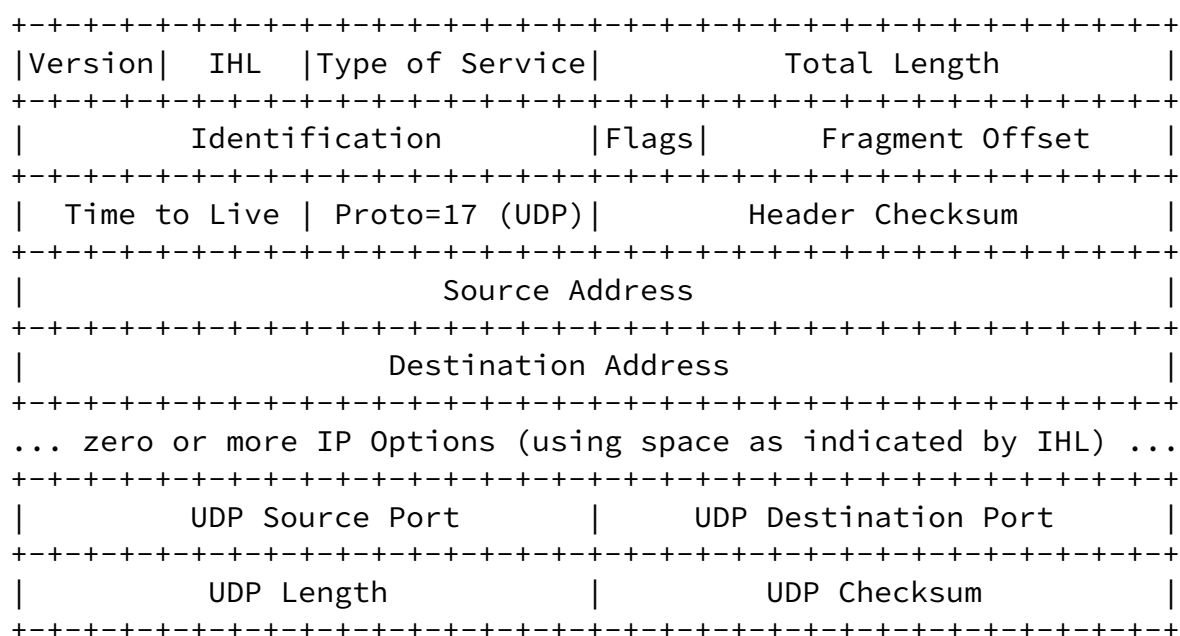


Figure 1 IPv4 datagram with UDP header

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 HDR 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 typical (and largest valid) value for UDP Length is:

$$\text{UDP_Length} = \text{IPv6_Payload_Length} - \text{sum}(\text{extension header lengths})$$

```

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|Version| Traffic Class |           Flow Label           |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           Payload Length           | Next Hdr   | Hop Limit   |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
...
|           Source Address (128 bits)           |
...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
...
|           Destination Address (128 bits)           |
...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
... zero or more IP Extension headers (each indicating size) ...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           UDP Source Port           |           UDP Destination Port           |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

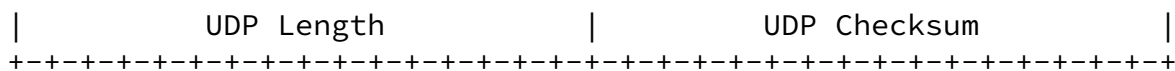


Figure 2 IPv6 datagram with UDP header

In both cases, the space available for the UDP packet is indicated by IP, either directly in the base header (for IPv4) or by adding information in the extensions (for IPv6). 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).

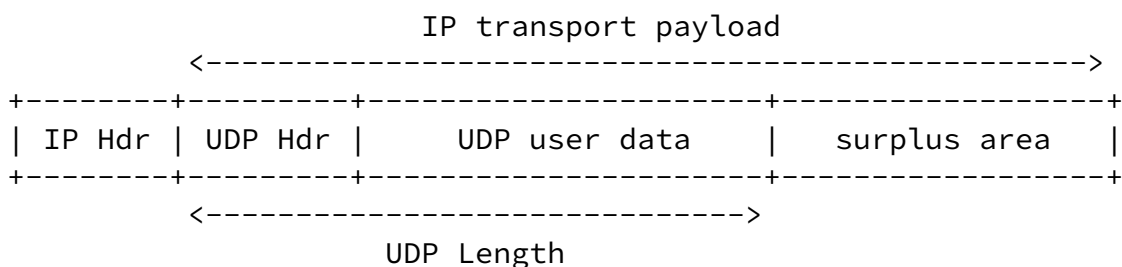


Figure 3 IP transport payload vs. UDP Length

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 [[RFC768](#)] (discussed further in [Section 16](#)). 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".

6. 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, using zeroes for alignment. The UDP option area can be used with any UDP payload length (including zero), 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.

The OCS contains an optional ones-complement sum that detects errors in the surplus area, which is not otherwise covered by the UDP checksum, as detailed in [Section 7](#).

The remainder of the surplus area consists of options defined using a TLV (type, length, and optional value) syntax similar to that of TCP [[RFC793](#)], as detailed in [Section 8](#). These options continue until the end of the surplus area or can end earlier using the EOL (end of list) option, followed by zeroes.

[7](#). The Option Checksum (OCS)

The Option Checksum (OCS) option is conventional Internet checksum [[RFC791](#)] 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.

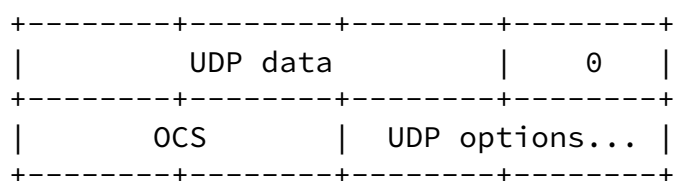


Figure 4 UDP OCS format, here using one zero for alignment

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 non-standard (i.e., non-option) uses of that area and accidental errors. It is not intended

to detect attacks, as discussed further in [Section 22](#).

The design enables traversal of errant middleboxes that incorrectly compute the UDP checksum over the entire IP payload [[Fa18](#)], 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, 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.

>> 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.

Like the UDP checksum, the OCS is optional under certain circumstances and contains zero when not used. UDP checksums can be zero for IPv4 [[RFC791](#)] and for IPv6 [[RFC8200](#)] when UDP payload 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 9.4](#)).

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

>> If 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 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 12](#)).

[8](#). UDP Options

UDP options are typically a minimum of two bytes in length as shown

in Figure 5, excepting only the one byte options "No Operation" (NOP) and "End of Options List" (EOL) described below.

```
+-----+-----+-----+
| Kind | Length | (remainder of option...)
+-----+-----+-----+
```

Figure 5 UDP option default format

The Kind field is always one byte. The Length field 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.

```
+-----+-----+-----+-----+
| Kind | 255 | Extended Length |
+-----+-----+-----+-----+
| (remainder of option...)
+-----+-----+-----+-----+
```

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 where rate-limiting permits.

>> 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 EOL and NOP MAY use either the default or extended option formats.

>> 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.

The following UDP options are currently defined:

| Kind | Length | Meaning |
|---------|----------|--|
| ----- | | |
| 0* | - | End of Options List (EOL) |
| 1* | - | No operation (NOP) |
| 2* | 6 | Alternate payload checksum (APC) |
| 3* | 10/12 | Fragmentation (FRAG) |
| 4* | 4 | Maximum datagram size (MDS) |
| 5* | 4 | Maximum reassembled datagram size (MRDS) |
| 6* | 6 | Request (REQ) |
| 7* | 6 | Response (RES) |
| 8 | 10 | Timestamps (TIME) |
| 9 | (varies) | Authentication (AUTH) |
| 10-126 | (varies) | UNASSIGNED (assignable by IANA) |
| 127 | (varies) | RFC 3692 -style experiments (EXP) |
| 128-191 | | RESERVED |
| 192 | (varies) | Encryption (UENC) |
| 193-253 | | UNASSIGNED-UNSAFE (assignable by IANA) |
| 254 | (varies) | RFC 3692 -style experiments (UEXP) |
| 255 | | RESERVED |

Options indicated by Kind values in the range 0..127 are known as SAFE options because they do not alter the UDP data payload and thus do not interfere with use of that data by legacy endpoints. Options indicated by Kind values in the range 192..254 are known as UNSAFE options because they do alter the UDP data payload and thus would interfere with legacy endpoints. UNSAFE option nicknames are expected to begin with "U", which should be avoided for safe option nicknames (see [Section 23](#)). Kind values 128-191 and 255 are RESERVED and not otherwise defined at this time.

>> RESERVED Kind values MUST NOT be assumed to be either SAFE nor UNSAFE until defined.

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 UDP payloads, thus is SAFE in actual use, as discussed further in [Section 9.4](#).

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 a "*" 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.

>> An endpoint supporting UDP options MUST treat unsupported options in the UNSAFE range as terminating all option processing.

>> All other SAFE options (without a "*") 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 SAFE options (i.e., in the same way a legacy receiver would). That includes options whose length does not indicate the specified value(s), as long as the length is not inherently invalid (i.e., smaller than 2 for the default and 4 for the extended formats).

>> UNSAFE options are used only in with the FRAG option, in a manner that prevents them from being silently ignored but passing the UDP payload to the user when not supported. This ensures their safe use in environments that might include legacy receivers (See [Section 10](#)).

>> Receivers supporting UDP options MUST silently drop all UDP options in a datagram containing an UNSAFE option when any UNSAFE option it contains is unknown. See [Section 10](#) for further discussion of UNSAFE options.

>> Except for NOP, EXP, and UEXP, each option SHOULD NOT occur more than once in a single UDP datagram. If an option other than these occurs more than once, a receiver MUST interpret only the first instance of that option and MUST ignore all others.

>> EXP and UEXP MAY occur more than once, but SHOULD NOT occur more than once using the same ExID (see Sections [9.10](#) and [10.2](#)).

>> Only the OCS and the AUTH and UENC options depend on the contents of the surplus area. AUTH and UENC are never used together, as UENC would serve both purposes. 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 a value dependent on the contents of the surplus area. Otherwise, interactions between those values, the OCS, and the AUTH and UENC options could be unpredictable.

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). The exception is only for lengths that imply a physical impossibility, e.g., smaller than two for conventional options and four for extended length options. Impossible lengths should indicate a malformed surplus area and all options silently discarded. Lengths other than those expected should result in safe options being ignored and skipped over, as with any other unknown safe option.

>> Option lengths MUST NOT exceed the IP length of the overall IP datagram. If this occurs, the options MUST be treated as malformed and all options dropped, and the event MAY be logged for diagnostics (logging SHOULD be rate limited).

>> "Must-support" options other than NOP and EOL MUST come before other options.

The requirement that must-support options come before others is intended to allow for endpoints to implement DOS protection, as discussed further in [Section 22](#).

[9](#). 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 10](#)). The following subsections describe safe options defined in this document.

[9.1](#). 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.

```

+-----+
| Kind=0 |
+-----+

```

Figure 7 UDP EOL option format

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

Touch

Expires September 26, 2022

[Page 13]

Internet-Draft

Transport Options for UDP

March 2022

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

>> All bytes in the surplus area after EOL MUST be set to zero on transmit.

>> Bytes after EOL in the surplus area MAY be checked as being zero on receipt but MUST be treated as zero regardless of their content and are not passed to the user (e.g., as part of the surplus area).

Requiring the post-option surplus area to be zero prevents side-channel uses of this area, requiring instead 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 [[Fa22](#)]).

[9.2](#). 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.

```

+-----+
| Kind=1 |
+-----+

```

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.

This issue is discussed further in [Section 22](#).

[9.3](#). Alternate Payload Checksum (APC)

The Alternate Payload Checksum (APC, Kind=2) option provides a stronger alternative to the checksum in the UDP header, using a 32-bit CRC of the conventional UDP user data payload only (excluding the IP pseudoheader, UDP header, and surplus area). It is an "alternate" to the UDP checksum that covers the user data – not to the OCS (the latter covers the surplus area only). Unlike the UDP checksum, APC does not include the IP pseudoheader or UDP header, thus it does not need to be updated by NATs when IP addresses or UDP

Touch

Expires September 26, 2022

[Page 14]

Internet-Draft

Transport Options for UDP

March 2022

ports are rewritten. Its purpose is to detect user data errors that the UDP checksum, when used, might not detect.

A CRC32c has been chosen because of its ubiquity and use in other Internet protocols, including iSCSI and SCTP. The option contains the CRC32c in network standard byte order, as described in [\[RFC3385\]](#).

```
+-----+-----+-----+-----+
| Kind=2 | Len=6  |   CRC32c...   |
+-----+-----+-----+-----+
|  CRC32c (cont.) |
+-----+-----+-----+-----+
```

Figure 9 UDP APC option format

When present, the APC always contains a valid CRC checksum. There are no reserved values, including the value of zero. If the CRC is zero, this must indicate a valid checksum (i.e., it does not indicate that the APC is not used; instead, the option would simply not be included if that were the desired effect).

APC does not protect the UDP pseudoheader; only the current UDP checksum provides that protection (when used). 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; because this is not the case, APC does not cover the pseudoheader.

>> UDP packets with incorrect APC checksums MUST be passed to the application by default, e.g., with a flag indicating APC failure.

Like all safe UDP options, APC needs to be silently ignored when failing by default, unless the receiver has been configured to do otherwise. Although all UDP option-aware endpoints support APC (being in the required set), this silently-ignored behavior ensures that option-aware receivers operate the same as legacy receivers unless overridden.

>> UDP packets with unrecognized APC lengths MUST be 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 as APC-like.

[9.4.](#) Fragmentation (FRAG)

The Fragmentation (FRAG, Kind=3) option supports UDP fragmentation and reassembly, which can be used to transfer UDP messages larger than limited by the IP receive MTU (EMTU_R [[RFC1122](#)]). FRAG includes a copy of the same UDP transport ports in each fragment, enabling them to traverse Network Address (and port) Translation (NAT) devices, in contrast to the behavior of IP fragments. 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. FRAG is designed similar to 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.

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

>> 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.

```

+-----+-----+-----+-----+
| Kind=3 | Len=10 |  Frag. Start  |
+-----+-----+-----+-----+
|                Identification                |
+-----+-----+-----+-----+
|  Frag. Offset  |
+-----+-----+

```

Figure 10 UDP non-terminal FRAG option format

In the non-terminal FRAG option format, Frag. Start 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 are applied to this UDP fragment. Non-terminal fragments never have options after the fragment.

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

The FRAG option does not need a "more fragments" bit 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 2 fragments, for a total of at least 3,000 bytes (see MRDS in [Section 9.6](#)).

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.

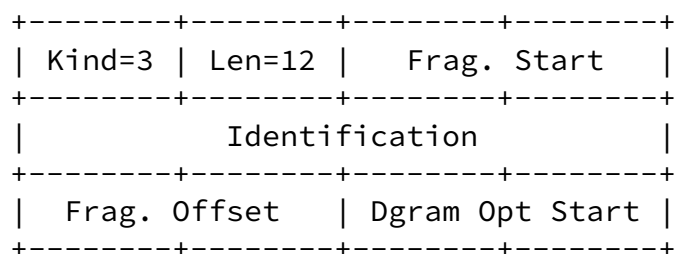


Figure 11 UDP terminal FRAG option format

The terminal FRAG option format adds a Datagram Option Start 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 after the original UDP datagram. In this variant, UDP options that apply to the reassembled datagram may occur after the terminal fragment data. UDP options that occur before the FRAG data are processed on the fragment; UDP options after the FRAG data are processed after reassembly, such that the reassembled data represents the original UDP user data. 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.

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

The Fragment Offset is 16 bits and indicates the location of the UDP payload fragment in bytes from the beginning of the original unfragmented payload. The option Len field indicates whether there are more fragments (Len=10) or no more fragments (Len=12).

>> 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 should be sent.

It should be noted that fragments may 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 SHOULD be no more than 2 minutes.

>> 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-socketpair 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.

Any per-datagram UDP options, if used, follow the FRAG option in the final fragment and would be included in the reassembled UDP packet. Processing of those options would commence after reassembly. This is especially important for UNSAFE options, which are interpreted only after FRAG.

In general, UDP packets are fragmented as follows:

1. Create a UDP packet with data and UDP options, which we will call "D". Note that the UDP options treat the data area as UDP user data and thus must follow that data.

Process these UDP options before the rest of the fragmentation steps below. Note that 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 the UDP checksum is non-zero).

2. Identify the desired fragment size, which we will call "S". This value should take into account the path MTU (if known) and allow space for per-fragment options.
3. Fragment "D" into chunks of size no larger than "S"-10 each, with one final chunk no larger than "S"-12. Note that all the non-FRAG options in step #1 need not be limited to the terminal fragment, i.e., the Dgram Opt. Start pointer can indicate the start of the original surplus area anywhere in the reassembled data.
4. For each chunk of "D" in step #3, create a zero-data UDP packet followed by the word-aligned OCS, the FRAG option, and any additional UDP options, followed by the FRAG data chunk.

The last chunk includes the non-FRAG options noted in step #1 after the end of the FRAG data. These UDP options apply to the reassembled user data as a whole when received.

5. Process the pre-reassembly UDP options of 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.

[9.5](#). Maximum Datagram Size (MDS)

The Maximum Datagram Size (MDS, Kind=4) option is a 16-bit hint of the largest unfragmented UDP packet that an endpoint believes can be

received. As with the TCP Maximum Segment Size (MSS) option [[RFC793](#)], the size indicated is the IP layer MTU decreased by the fixed IP and UDP headers only [[RFC6691](#)]. The space needed for IP and

UDP options need 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; it does not indicate a known path MTU and thus MUST NOT be used to limit transmissions.

```

+-----+-----+-----+-----+
| Kind=4 | Len=4  |      MDS size      |
+-----+-----+-----+-----+

```

Figure 12 UDP MDS option format

The UDP MDS option MAY be used as a hint for path MTU discovery [RFC1191][RFC8201], but this may be difficult because of known issues with ICMP blocking [RFC2923] as well as UDP lacking automatic retransmission. It 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 9.6](#)), e.g., when EMTU_R is larger than the required minimums (576 for IPv4 [RFC791] and 1500 for IPv6 [RFC8200]). It can also be used with DPLPMTUD [RFC8899] to provide a hint to maximum DPLPMTU, though it MUST NOT prohibit transmission of larger UDP packets (or fragments) used as DPLPMTU probes.

[9.6](#). Maximum Reassembled Datagram Size (MRDS)

The Maximum Reassembled Segment Size (MRDS, Kind=5) option is a 16-bit indicator of the largest reassembled UDP segment that can be received. MRDS is the UDP equivalent of IP's EMTU_R but the two are not related [RFC1122]. Using the FRAG option ([Section 9.4](#)), UDP packets can be transmitted as transport fragments, each in their own (presumably not fragmented) IP datagram and be reassembled at the UDP layer.

```

+-----+-----+-----+-----+
| Kind=5 | Len=4  |      MRDS size      |
+-----+-----+-----+-----+

```

Figure 13 UDP MRDS option format

>> Endpoints supporting UDP options MUST support a local MRDS of at least 3,000 bytes.

9.7. Echo request (REQ) and echo response (RES)

The echo request (REQ, Kind=6) and echo response (RES, Kind=7) options provide a means for UDP options to be used to provide UDP packet-level acknowledgements. One such use is described as part of the UDP options variant of packetization layer path MTU discovery (PLPMTUD) [Fa22]. The options both have the format indicated in Figure 14, in which the token has no internal structure or meaning.

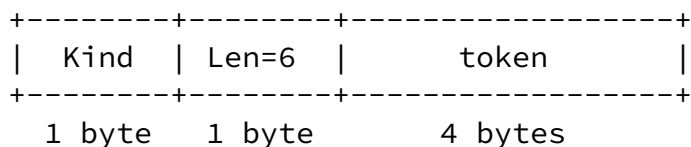


Figure 14 UDP REQ and RES options format

Each of these option kinds appears at most once in each UDP packet, as with other options. Note also that the FRAG option is not used when sending DPLPMTUD probes to determine a PLPMTU [Fa22].

9.8. Timestamps (TIME)

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 round trip time (RTT) between hosts. For UDP, this RTT can be useful for establishing UDP fragment reassembly timeouts or transport-layer rate-limiting [RFC8085].

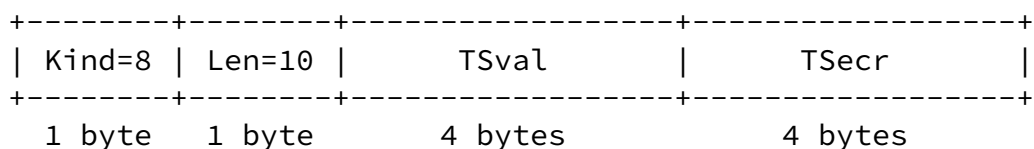


Figure 15 UDP TIME option format

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, TS Value 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 nonzero 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 be:

- o Values are monotonic and non-decreasing except for anticipated number-space rollover events
- o Values should "increase" (allowing for rollover) according to a typical 'tick' time
- o A request is defined as TSval being non-zero and a reply is defined as TSecr being non-zero.
- o A receiver should always respond 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.

[9.9](#). Authentication (AUTH)

The Authentication (AUTH, Kind=9) option is intended to allow UDP to provide a similar type of authentication as the TCP Authentication Option (TCP-AO) [[RFC5925](#)]. AUTH covers the UDP user data. AUTH supports NAT traversal in a similar manner as TCP-AO [[RFC6978](#)]. Figure 16 shows the UDP AUTH format, whose contents are identical to that of the TCP-AO option.

```
+-----+-----+-----+-----+
| Kind=9 | Len   | TCP-AO fields...|
+-----+-----+-----+-----+
|          TCP-AO fields (con't)...          |
```


+-----+-----+-----+-----+

Figure 16 UDP AUTH option format

Touch

Expires September 26, 2022

[Page 22]

Internet-Draft

Transport Options for UDP

March 2022

Like TCP-AO, AUTH is not negotiated in-band. Its use assumes both endpoints have populated Master Key Tuples (MKTs), used to exclude non-protected traffic.

TCP-AO generates unique traffic keys from a hash of TCP connection parameters. UDP lacks a three-way handshake to coordinate connection-specific values, such as TCP's Initial Sequence Numbers (ISNs) [[RFC793](#)], thus AUTH's Key Derivation Function (KDF) uses zeroes as the value for both ISNs. This means that the AUTH reuses keys when socket pairs are reused, unlike TCP-AO.

>> UDP packets with incorrect AUTH HMACs MUST be passed to the application by default, e.g., with a flag indicating AUTH failure.

Like all non-UNSAFE UDP options, AUTH needs to be silently ignored when failing. This silently-ignored behavior ensures that option-aware receivers operate the same as legacy receivers unless overridden.

In addition to the UDP user data (which is always included), AUTH can be configured to either include or exclude the surplus area, in a similar way as can TCP-AO optionally exclude TCP options. When UDP options are covered, the OCS value and AUTH (and later, UENC) hash areas are zeroed before computing the AUTH hash. It is important to consider that options not yet defined might yield unpredictable results if not confirmed as supported, e.g., if they were to contain other hashes or checksums that depend on the surplus area contents. This is why such dependencies are not permitted except as defined for the OCS and the AUTH (and later, UENC) option.

Similar to TCP-AO-NAT, AUTH (and later, UENC) can be configured to support NAT traversal, excluding (by zeroing out) one or both of the UDP ports and corresponding IP addresses [[RFC6978](#)].

[9.10](#). Experimental (EXP)

The Experimental option (EXP, Kind=127) is reserved for experiments [[RFC3692](#)]. Only one such 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](#)].

```

+-----+-----+-----+-----+
| Kind=127 |  Len  |      UDP ExID      |
+-----+-----+-----+-----+
| (option contents, as defined)... |
+-----+-----+-----+-----+

```

Figure 17 UDP EXP option format

>> The length of the experimental option MUST be at least 4 to account for the Kind, Length, and the minimum 16-bit UDP ExID identifier (similar to TCP ExIDs [[RFC6994](#)]).

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

```

+-----+-----+-----+-----+
| Kind=127 |  255  |      Extended Length  |
+-----+-----+-----+-----+
|      UDP ExID.      | (option contents...) |
+-----+-----+-----+-----+

```

Figure 18 UDP EXP option format

Assigned UDP experimental IDs (ExIDs) assigned from a single registry managed by IANA (see [Section 23](#)). Assigned ExIDs can be used in either the EXP or UEXP options (see [Section 10.2](#) for the latter).

[10](#). 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

Touch

Expires September 26, 2022

[Page 24]

Internet-Draft

Transport Options for UDP

March 2022

datagram includes an UNSAFE option whose UKind is not supported. Note that this still results in the receipt of a zero-length UDP datagram.

[10.1](#). UNSAFE Encryption (UENC)

UNSAFE encryption (UENC, Kind=192) has the same format as AUTH ([Section 9.9](#)), except that it encrypts (modifies) the user data. It provides a similar encryption capability as TCP-AO-ENC, in a similar manner [[To18](#)]. Its fields, coverage, and processing are the same as for AUTH, except that UENC encrypts only the user data, although it can (optionally) depend on the surplus area (with certain fields zeroed, as per AUTH, e.g., providing authentication over the surplus area). Like AUTH, UENC can be configured to be compatible with NAT traversal.

[10.2](#). UNSAFE Experimental (UEXP)

The UNSAFE Experimental option (UEXP, Kind=254) 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.

[11](#). Rules for designing new options

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

>> New options MUST NOT modify other option content.

>> 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 UDP option capable receivers.

>> New options MUST NOT declare their order relative to other options, whether new or old.

>> At the sender, new options MUST NOT modify UDP packet content anywhere except within their option field, excepting only those

Touch

Expires September 26, 2022

[Page 25]

Internet-Draft

Transport Options for UDP

March 2022

contained within the UNSAFE option; areas that need to remain unmodified include the IP header, IP options, the UDP user data, and the 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 intend optionally for modification of any UDP options, including their new areas, in transit.

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

- o >> Only FRAG and UNSAFE options are permitted to modify the UDP body.

The following recommendation helps enable efficient zero-copy processing:

- o >> FRAG SHOULD be the first option, when present.

[12](#). 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.

>> All options except UNSAFE options MUST result in the UDP user data being passed to the application layer, regardless of whether all options are processed, 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 its options as follows (in order):

```
if the UDP checksum fails then
    silently drop the entire UDP packet (per RFC1122)
if the UDP checksum passes then
    if OCS != 0 and fails or is zero when UDP CS != 0 then
        deliver the UDP user data but ignore other options
        (this is required to emulate legacy behavior)
    if OCS is nonzero and passes or is zero 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 as used only inside the FRAG area ensures that the resulting UDP data will be silently dropped in both legacy and options-aware receivers. 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.

I.e., 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) should the receiver require it being present and correct.

I.e., for all options:

- o if the option is not required by the receiver, then UDP packets missing the option are accepted.
- o 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.
- o 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.

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).

13. UDP API Extensions

UDP currently specifies an application programmer interface (API), summarized as follows (with Unix-style command as an example) [[RFC768](#)]:

- o Method to create new receive ports
 - o E.g., `bind(handle, recvaddr(optional), recvport)`

- o Receive, which returns data octets, source port, and source address
 - o E.g., `recvfrom(handle, srcaddr, srcport, data)`
- o Send, which specifies data, source and destination addresses, and source and destination ports
 - o E.g., `sendto(handle, destaddr, destport, data)`

This API is extended to support options as follows:

- o Extend the method to create receive ports to include per-packet and per-fragment receive options that are required as indicated by the application. Datagrams not containing these required options **MUST** be silently dropped and **MAY** be logged. This includes a minimum datagram length, such that the options list ends in EOL and additional space is zero-filled as needed.
- o WG QUESTION: DO WE ALSO WANT A MIN FRAG SIZE? OR MAX?
- o 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.
- o WG QUESTION: SHOULD WE ACCUMULATE THOSE OPTIONS? OR DISCARD THEM?
- o 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.

Examples of API instances for Linux and FreeBSD are provided in [Appendix A](#), to encourage uniform cross-platform implementations.

14. 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 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 intended for use only by the transport endpoints. They are no more (or less) appropriate to be modified in-transit than any other portion of the transport datagram.

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

[15.](#) 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.

16. Interactions with Legacy Devices

It has always been permissible for the UDP Length to be inconsistent with the IP transport payload length [[RFC768](#)]. 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 [[RFC768](#)] [[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., CheckPoint, use a default "relaxed UDP length verification" to avoid falsely interpreting this inconsistency as an attack.

17. 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 MUST allow for message reordering and loss.

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.

18. UDP Option State Caching

Some TCP connection parameters, stored in the TCP Control Block, 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 may have similar benefit 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.

19. Updates to [RFC 768](#)

This document updates [RFC 768](#) as follows:

- o 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.
- o This document extends the UDP API to support the use of UDP options.

[20](#). 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 [[RFC768](#)] [[RFC1122](#)].

Teredo extensions (TE) define use of a similar difference between these lengths for trailers [[RFC6081](#)]. TE defines the UDP length pointing beyond (larger) than the location indicated by the IP length rather than shorter (as used herein):

"..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)"

As a result, UDP options are not compatible with TE, but that is also why this document does not update TE. Additionally, it is not at all clear how TE operates, as it requires network processing of the UDP length field to understand the total message including TE trailers.

TE updates Teredo NAT traversal [[RFC4380](#)]. The NAT traversal document defined "consistency" of UDP length and IP length as:

"An IPv6 packet is deemed valid if it conforms to [[RFC2460](#)]: the protocol identifier should indicate an IPv6 packet and the payload length should be consistent with the length of the UDP datagram in which the packet is encapsulated."

IPv6 is clear on the meaning of this consistency, in which the pseudoheader used for UDP checksums is based on the UDP length, not inferred from the IP length, using the same text in the current specification [[RFC8200](#)]:

"The Upper-Layer Packet Length in the pseudo-header is the length of the upper-layer header and data (e.g., TCP header plus TCP data). Some upper-layer protocols carry their own length information (e.g., the Length field in the UDP header); for such protocols, that is the length used in the pseudo-header."

This document is consistent 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

Touch

Expires September 26, 2022

[Page 32]

Internet-Draft

Transport Options for UDP

March 2022

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](#)) would cause the UDP header to remain uncompressed. Upon receipt of a compressed UDP header, Section A.1.3 of that document 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., Sec. 10.10 of Static Context Header Compression [[RFC8724](#)].

[21](#). Multicast Considerations

UDP options are primarily intended for unicast use. Using these options over multicast 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 group confirm support for the options in use.

[22](#). 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 in this section further.

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. 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.

UDP options are not covered by DTLS (datagram transport-layer

security). Despite the name, neither TLS [[RFC8446](#)] (transport layer security, for TCP) nor DTLS [[RFC6347](#)] (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](#)]) or in UDP by the Authentication (AUTH) option ([Section 9.9](#)) and UNSAFE Encryption

(UENC) option ([Section 10](#)). Transport headers are also protected as payload when using IP security (IPsec) [[RFC4301](#)].

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

>> Implementations concerned with the potential for this vulnerability MAY implement only the required UDP options and MAY also limit processing of TLVs, either in number of non-padding options or total length, or both. The number of non-zero TLVs allowed in such cases MUST be at least 8.

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

UDP security should never 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 may be dropped either if that default is 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).

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 may be useful to assume a reasonable minimum fragment size, e.g., that non-terminal fragments should never be smaller than 500 bytes.

[23.](#) IANA Considerations

Upon publication, IANA is hereby requested to create a new registry for UDP Option Kind numbers, similar to that for TCP Option Kinds. Initial values of this registry are as listed in [Section 8](#). Additional values in this registry are to be assigned from the UNASSIGNED values in [Section 8](#) by IESG Approval or Standards Action

Touch

Expires September 26, 2022

[Page 34]

Internet-Draft

Transport Options for UDP

March 2022

[[RFC8126](#)]. Those assignments are subject to the conditions set forth in this document, particularly (but not limited to) those in [Section 11](#).

Although option nicknames are not used in-band, IANA should require UNSAFE safe option values to commence with the letter "U" and avoid that letter as commencing safe options.

Upon publication, IANA is hereby requested to create a new registry for UDP Experimental Option Experiment Identifiers (UDP ExIDs) for use in a similar manner as TCP ExIDs [[RFC6994](#)]. UDP ExIDs can be used in either (or both) the EXP or UEXP options. This registry is initially empty. Values in this registry are to be assigned by IANA using first-come, first-served (FCFS) rules [[RFC8126](#)]. Options using these ExIDs are subject to the same conditions as new options, i.e., they too are subject to the conditions set forth in this document, particularly (but not limited to) those in [Section 11](#).

[24.](#) References

[24.1.](#) Normative References

- [Fa22] Fairhurst, G., T. Jones, "Datagram PLPMTUD for UDP Options," [draft-ietf-tsvwg-udp-options-dplpmtud](#), Feb. 2022.
- [RFC768] Postel, J., "User Datagram Protocol," [RFC 768](#), August 1980.

- [RFC791] Postel, J., "Internet Protocol," [RFC 791](#), Sept. 1981.
- [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts -- Communication Layers," [RFC 1122](#), Oct. 1989.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," [BCP 14](#), [RFC 2119](#), March 1997.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in [RFC 2119](#) Key Words," [RFC 2119](#), May 2017.

[24.2](#). Informative References

- [Fa18] Fairhurst, G., T. Jones, R. Zullo, "Checksum Compensation Options for UDP Options", [draft-fairhurst-udp-options-cco](#), Oct. 2018.

| | | |
|-------|----------------------------|-----------|
| Touch | Expires September 26, 2022 | [Page 35] |
|-------|----------------------------|-----------|

| | | |
|----------------|---------------------------|------------|
| Internet-Draft | Transport Options for UDP | March 2022 |
|----------------|---------------------------|------------|

- [Hi15] Hildebrand, J., B. Trammel, "Substrate Protocol for User Datagrams (SPUD) Prototype," [draft-hildebrand-spud-prototype-03](#), Mar. 2015.
- [RFC793] Postel, J., "Transmission Control Protocol" [RFC 793](#), September 1981.
- [RFC1071] Braden, R., D. Borman, C. Partridge, "Computing the Internet Checksum," [RFC 1071](#), Sept. 1988.
- [RFC1191] Mogul, J., S. Deering, "Path MTU discovery," [RFC 1191](#), November 1990.
- [RFC2923] Lahey, K., "TCP Problems with Path MTU Discovery," [RFC 2923](#), September 2000.
- [RFC3095] Bormann, C. (Ed), et al., "RObust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed," [RFC 3095](#), July 2001.
- [RFC3385] Sheinwald, D., J. Satran, P. Thaler, V. Cavanna, "Internet Protocol Small Computer System Interface (iSCSI) Cyclic Redundancy Check (CRC)/Checksum Considerations," [RFC 3385](#),

Sep. 2002.

- [RFC3692] Narten, T., "Assigning Experimental and Testing Numbers Considered Useful," [RFC 3692](#), Jan. 2004.
- [RFC3828] Larzon, L-A., M. Degermark, S. Pink, L-E. Jonsson (Ed.), G. Fairhurst (Ed.), "The Lightweight User Datagram Protocol (UDP-Lite)," [RFC 3828](#), July 2004.
- [RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", [RFC 4301](#), Dec. 2005.
- [RFC4340] Kohler, E., M. Handley, and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", [RFC 4340](#), March 2006.
- [RFC4380] Huitema, C., "Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)," [RFC 4380](#), Feb. 2006.
- [RFC4960] Stewart, R. (Ed.), "Stream Control Transmission Protocol", [RFC 4960](#), September 2007.
- [RFC5925] Touch, J., A. Mankin, R. Bonica, "The TCP Authentication Option," [RFC 5925](#), June 2010.

| | | |
|-------|----------------------------|-----------|
| Touch | Expires September 26, 2022 | [Page 36] |
|-------|----------------------------|-----------|

| | | |
|----------------|---------------------------|------------|
| Internet-Draft | Transport Options for UDP | March 2022 |
|----------------|---------------------------|------------|

- [RFC6081] Thaler, D., "Teredo Extensions," [RFC 6081](#), Jan 2011.
- [RFC6347] Rescorla, E., N. Modadugu, "Datagram Transport Layer Security Version 1.2," [RFC 6347](#), Jan. 2012.
- [RFC6691] Borman, D., "TCP Options and Maximum Segment Size (MSS)," [RFC 6691](#), July 2012.
- [RFC6935] Eubanks, M., P. Chimento, M. Westerlund, "IPv6 and UDP Checksums for Tunneled Packets," [RFC 6935](#), April 2013.
- [RFC6978] Touch, J., "A TCP Authentication Option Extension for NAT Traversal", [RFC 6978](#), July 2013.
- [RFC6994] Touch, J., "Shared Use of Experimental TCP Options," [RFC 6994](#), Aug. 2013.
- [RFC7323] Borman, D., R. Braden, V. Jacobson, R. Scheffenegger

(Ed.), "TCP Extensions for High Performance," [RFC 7323](#), Sep. 2014.

- [RFC8085] Eggert, L., G. Fairhurst, G. Shepherd, "UDP Usage Guidelines," [RFC 8085](#), Feb. 2017.
- [RFC8126] Cotton, M., B. Leiba, T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs," [RFC 8126](#), June 2017.
- [RFC8200] Deering, S., R. Hinden, "Internet Protocol Version 6 (IPv6) Specification," [RFC 8200](#), Jul. 2017.
- [RFC8201] McCann, J., S. Deering, J. Mogul, R. Hinden (Ed.), "Path MTU Discovery for IP version 6," [RFC 8201](#), Jul. 2017.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3," [RFC 8446](#), Aug. 2018.
- [RFC8504] Chown, T., J. Loughney, T. Winters, "IPv6 Node Requirements," [RFC 8504](#), Jan. 2019.
- [RFC8724] Minaburo, A., L. Toutain, C. Gomez, D. Barthel, JC., "SCHC: Generic Framework for Static Context Header Compression and Fragmentation," [RFC 8724](#), Apr. 2020.
- [RFC8899] Fairhurst, G., T. Jones, M. Tuxen, I. Rungeler, T. Volker, "Packetization Layer Path MTU Discovery for Datagram Transports," [RFC 8899](#), Sep. 2020.

Touch

Expires September 26, 2022

[Page 37]

Internet-Draft

Transport Options for UDP

March 2022

- [RFC9040] Touch, J., M. Welzl, S. Islam, "TCP Control Block Interdependence," [RFC 9040](#), Jul. 2021.
- [RFC9187] Touch, J., "Sequence Number Extension for Windowed Protocols," [RFC 9187](#), Jan. 2022.
- [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>.
- [To18] Touch, J., "A TCP Authentication Option Extension for Payload Encryption," [draft-touch-tcp-ao-encrypt](#), Jul.

2018.

[25.](#) Acknowledgments

This work benefitted from feedback from Erik Auerswald, Bob Briscoe, Ken Calvert, Ted Faber, Gorrry Fairhurst (including OCS for misbehaving middlebox traversal), C. M. Heard (including combining previous FRAG and LITE options into the new FRAG), Tom Herbert, Mark Smith, 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.

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

Authors' Addresses

Joe Touch
Manhattan Beach, CA 90266 USA

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

| | | |
|-------|----------------------------|-----------|
| Touch | Expires September 26, 2022 | [Page 38] |
|-------|----------------------------|-----------|

| | | |
|----------------|---------------------------|------------|
| Internet-Draft | Transport Options for UDP | March 2022 |
|----------------|---------------------------|------------|

[Appendix A.](#) Implementation Information

The following information is provided to encourage interoperable API implementations.

System-level variables (sysctl):

| Name | default | meaning |
|------|---------|---------|
|------|---------|---------|

```

-----
net.ipv4.udp_opt      0      UDP options available
net.ipv4.udp_opt_ocs  1      Default use OCS
net.ipv4.udp_opt_apc  0      Default include APC
net.ipv4.udp_opt_frag 0      Default fragment
net.ipv4.udp_opt_mds  0      Default include MDS
net.ipv4.udp_opt_mrds 0      Default include MRDS
net.ipv4.udp_opt_req  0      Default include REQ
net.ipv4.udp_opt_resp 0      Default include RES
net.ipv4.udp_opt_time 0      Default include TIME
net.ipv4.udp_opt_auth 0      Default include AUTH
net.ipv4.udp_opt_exp  0      Default include EXP
net.ipv4.udp_opt_uenc 0      Default include UENC
net.ipv4.udp_opt_uexp 0      Default include UEXP

```

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_UENC      Enable UDP UENC option
UDP_OPT_UEXP      Enable UDP UEXP option

```

Send/sendto parameters:

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

```

Name              Initial value
-----
opts_enabled      net.ipv4.udp_opt

```

ocs_enabled net.ipv4.udp_opt_ocs

The following option is included for debugging purposes, and MUST NOT 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 |

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 |

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

| Name | Initial value |
|--------------|-----------------------|
| ----- | |
| junk_enabled | net.ipv4.udp_opt_junk |
| junk_value | 0xABCD |
| junk_len | 4 |

