Transport Options for UDP
draft-ietf-tsvwg-udp-options-09.txt

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79. This document may not be modified, and derivative works of it may not be created, except to format it for publication as an RFC or to translate it into languages other than English.

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

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

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

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

This Internet-Draft will expire on May 25, 2021.

Copyright Notice

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.

Table of Contents

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. 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].

These 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. This document is intended to provide an out-of-band option area as an alternative to the in-band mechanism currently proposed [Hi15].

UDP is one of the most popular protocols that lacks space for options [RFC768]. The UDP header was intended to be a minimal addition to IP, providing only ports and a data checksum for

protection. This document extends UDP to provide a trailer area for options located after the UDP data payload.

4. The UDP Option Area

The UDP transport header includes demultiplexing and service identification (port numbers), a 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 (see also discussion in Section 11).

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:

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

For IPv6, the IP Payload Length field indicates the datagram 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:

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

In both cases, the space available for the UDP transport protocol data unit is indicated by IP, either completely in the base header (for IPv4) or adding information in the extensions (for IPv6). In either case, this document will refer to this available space as the "IP transport payload".

```
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |Version|  IHL  |Type of Service|          Total Length         |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |         Identification        |Flags|      Fragment Offset    |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |  Time to Live | Proto=17 (UDP)|        Header Checksum        |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                       Source Address                          |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                    Destination Address                        |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      ... zero or more IP Options (using space as indicated by IHL) ...
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |         UDP Source Port       |     UDP Destination Port      |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |          UDP Length           |         UDP Checksum          |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1 IPv4 datagram with UDP transport payload

```
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |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      |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |          UDP Length           |         UDP Checksum          |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2 IPv6 datagram with UDP 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).

```
                         IP transport payload
             <------------------------------------------------->
       +--------+---------+----------------------+------------------+
       | IP Hdr | UDP Hdr |     UDP user data    |   surplus area   |
       +--------+---------+----------------------+------------------+
             <------------------------------>
                        UDP Length
```

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. It is important to note that this is not a requirement of UDP [RFC768] (discussed further in Section 11). UDP-Lite used the difference in these pointers to indicate the partial coverage of the UDP Checksum, such that the UDP user data, UDP header, and UDP pseudoheader (a subset of the IP header) are covered by the UDP checksum but additional user data in the surplus area is not covered [RFC3828]. This document uses the surplus area for UDP transport options.

The UDP option area is thus defined as the location between the end of the UDP payload and the end of the IP datagram as a trailing options area. This area can occur 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 offset".

UDP options are defined using a TLV (type, length, and optional value) syntax similar to that of TCP [RFC793]. They are typically a minimum of two bytes in length as shown in Figure 4, excepting only the one byte options "No Operation" (NOP) and "End of Options List" (EOL) described below.

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

Figure 4 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 5. The Extended Length field is a 16-bit field in network standard byte order.

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

Figure 5 UDP option extended default format

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

>> 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 and default format MUST be treated as an error. Such errors call into question the remainder of the option area and thus MUST result in all UDP options being silently discarded.

>> Any UDP option whose length is only smaller than 255 MUST always use the UDP option default format shown in Figure 4, excepting only EOL and NOP.

>> Any UDP option whose length can be larger than 254 MUST always use the UDP option extended default format shown in Figure 5, including UNSAFE and EXP.

I.e., a UDP option always uses only the default format or the extended default format, depending on whether its length is only ever smaller than 255 or not.

Others have considered using values of the UDP Length that is larger than the IP transport payload as an additional type of signal. Using a value smaller than the IP transport payload is expected to be backward compatible with existing UDP implementations, i.e., to deliver the UDP Length of user data to the application and silently ignore the additional surplus area data. Using a value larger than the IP transport payload would either be considered malformed (and be silently dropped) or could cause buffer overruns, and so is not considered silently and safely backward compatible. Its use is thus out of scope for the extension described in this document.

>> UDP options MUST be interpreted in the order in which they occur in the UDP option area.

5. UDP Options

The following UDP options are currently defined:

Kind	Length	Meaning
0*	-	End of Options List (EOL)
1*	-	No operation (NOP)
2*	3	Option checksum (OCS)
3*	6	Alternate checksum (ACS)
4*	10/12	Fragmentation (FRAG)
5*	4	Maximum segment size (MSS)
6*	(varies)	Unsafe to ignore (UNSAFE) options
7	10	Timestamps (TIME)
8	(varies)	Authentication and Encryption (AE)
9	6	Request (REQ)
10	6	Response (RES)
11-126	(varies)	UNASSIGNED (assignable by IANA)
127-253		RESERVED
254	(varies)	RFC 3692-style experiments (EXP)
255		RESERVED

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, OCS, ACS, FRAG, MSS, and UNSAFE. This includes both recognizing and being able to generate these options if configured to do so. These are called "must-support" options.

>> All other options (without a "*") MAY be implemented, and their use SHOULD be determined either out-of-band or negotiated.

>> Receivers supporting UDP options MUST silently ignore unknown options except UNSAFE. That includes options whose length does not indicate the specified value(s).

>> Receivers supporting UDP options MUST silently drop the entire datagram containing an UNSAFE option when any UNSAFE option it contains is unknown. See Section 5.7 for further discussion of UNSAFE options.

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

>> Only the OCS and AE options depend on the contents of the option area. AE is always computed as if the AE hash and OCS checksum are zero; OCS is always computed as if the OCS checksum is zero and after the AE hash has been computed. Future options MUST NOT be defined as having a value dependent on the contents of the option area. Otherwise, interactions between those values, OCS, and AE could be unpredictable.

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

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

>> Options with fixed lengths MUST use the default option format.

>> Options with variable lengths MUST use the default option format where their total length is 254 bytes or less.

>> Options using the extended option format MUST indicate extended lengths of 255 or higher; smaller extended length values MUST be treated as an error.

>> "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 17.

5.1. End of Options List (EOL)

The End of Options List (EOL) option indicates that there are no more options. It is used to indicate the end of the list of options without needing to pad the options to fill all available option space.

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

Figure 6 UDP EOL option format

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

>> All bytes in the surplus area after EOL MUST be zero. If these bytes are non-zero, the entire surplus area MUST be silently ignored and only the UDP data passed to the user with an adjusted UDP length to indicate that no options were present.

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 for such padding to increase the packet length without affecting the payload length, e.g., for UDP DPLPMTUD [Fa20].

5.2. No Operation (NOP)

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

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

Figure 7 UDP NOP option format

>> If options longer than one byte are used, NOP options SHOULD be used at the beginning of the UDP options area to achieve alignment as would be more efficient for active (i.e., non-NOP) options.

>> Segments SHOULD NOT use more than three consecutive NOPs. NOPs are intended to assist with alignment, not other padding or fill.

[NOTE: Tom Herbert suggested we declare "more than 3 consecutive NOPs" a fatal error to reduce the potential of using NOPs as a DOS attack, but IMO there are other equivalent ways (e.g., using RESERVED or other UNASSIGNED values) and the "no more than 3" creates its own DOS vulnerability)

5.3. Option Checksum (OCS)

The Option Checksum (OCS) option is conventional Internet checksum [RFC791] that covers all of the surplus area and a pseudoheader composed of the 16-bit length of the surplus area (Figure 8). The primary purpose of OCS is to detect non-standard (i.e., non-option) uses of that area. The surplus area pseudoheader is included to enable traversal of errant middleboxes that incorrectly compute the

UDP checksum over the entire IP payload rather than only the UDP payload [Fa18].

The OCS is calculated by computing the Internet checksum over the surplus area and surplus length pseudoheader. The OCS protects the option area from errors in a similar way that the UDP checksum protects the UDP user data (when not zero).

```
                        +--------+--------+
                        | surplus length  |
                        +--------+--------+
```

Figure 8 UDP surplus length pseudoheader

```
                    +--------+--------+--------+
                    | Kind=2 |     checksum    |
                    +--------+--------+--------+
```

Figure 9 UDP OCS option format

>> The OCS MUST be included when the UDP checksum is nonzero and UDP options are present.

>> When present, the OCS SHOULD occur as early as possible, preceded by only NOP options for alignment and the FRAG option if present.

>> OCS MUST be half-word coordinated with the start of the UDP options area and include the surplus length pseudoheader similarly coordinated with the start of UDP Header.

This Internet checksum is computed over the surplus area (including EOL, if present) prefixed by the surplus length pseudoheader (Figure 8) and then adjusting the result before storing it into the OCS checksum field. If the OCS checksum field is aligned to the start of the options area, then the checksum is inserted as-is, otherwise the checksum bytes are swapped before inserting them into the field. The effect of this "coordination" is the same is if the checksum were computed as if the surplus area and pseudoheader were aligned to the UDP header.

This feature is intended to potentially help the UDP options traverse devices that incorrectly attempt to checksum the surplus area (as originally proposed as the Checksum Compensation Option, i.e., CCO [Fa18]).

The OCS covers the UDP option area as formatted for transmission and immediately upon reception.

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

>> UDP 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 segment's socket pair.

As a reminder, use of the UDP checksum is optional when the UDP checksum is zero. When not used, the OCS is assumed to be "correct" for the purpose of accepting UDP packets at a receiver (see Section 7).

The OCS is intended to check for accidental errors, not for attacks.

5.4. Alternate Checksum (ACS)

The Alternate Checksum (ACS) option provides a stronger alternative to the checksum in the UDP header, using a 32-bit CRC of the conventional UDP payload only (excluding the IP pseudoheader, UDP header, and surplus area). It is an "alternate" to the UDP checksum (covering the UDP payload) - not the OCS (the latter covers the surplus area) Unlike the UDP checksum, ACS does not include the IP pseudoheader or UDP header, thus it does not need to be updated by NATs when IP addresses or UDP ports are rewritten. Its purpose is to detect UDP payload 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=3 | Len=6  |    CRC32c...    |
+--------+--------+--------+--------+
|  CRC32c (cont.) |
+--------+--------+
```

Figure 10   UDP ACS option format

When present, the ACS 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 ACS is not used; instead, the option would simply not be included if that were the desired effect).

ACS does not protect the UDP pseudoheader; only the current UDP checksum provides that protection (when used). ACS 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, ACS does not cover the pseudoheader.

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

Like all non-UNSAFE UDP options, ACS need to be silently ignored when failing. Although all UDP option-aware endpoints support ACS (being in the required set), this silently-ignored behavior ensures that option-aware receivers operate the same as legacy receivers unless overridden.

5.5. Fragmentation (FRAG)

The Fragmentation option (FRAG) combines properties of IP fragmentation and the UDP Lite transport protocol [RFC3828]. FRAG provides transport-layer fragmentation and reassembly in which each fragment includes a copy of the same UDP transport ports, enabling the fragments to traverse Network Address (and port) Translation (NAT) devices, in contrast to the behavior of IP fragments. FRAG also allows the UDP checksum to cover only a prefix of the UDP data payload, to avoid repeated checksums of data prior to reassembly.

The Fragmentation (FRAG) 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]). It is typically used with the UDP MSS option 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 MUST come first in the UDP options list.

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

Legacy receivers interpret FRAG messages as zero-length payload 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 packet itself were a signal.

The FRAG option has two formats; non-terminal fragments use the shorter variant (Figure 11) and terminal fragments use the longer (Figure 12). The latter includes stand-alone fragments, i.e., when data is contained in the FRAG option but reassembly is not required.

```
                +--------+--------+--------+--------+
                | Kind=4 | Len=10 |      Offset     |
                +--------+--------+--------+--------+
                |           Identification          |
                +--------+--------+--------+--------+
                |  Frag. Offset   |
                +--------+--------+
```

Figure 11   UDP non-terminal FRAG option format

The FRAG option does not need a "more fragments" bit because it provides the same indication by using the longer, 12-byte variant, which also includes an Internet checksum over the entire reassembled UDP payload (omitting the IP pseudoheader and UDP header, as well as UDP options), as shown in Figure 12.

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

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.

>> The reassembly checksum SHOULD be used, but MAY be unused in the same situations when the UDP checksum is unused (e.g., for transit tunnels or applications that have their own integrity checks [RFC8200]), and by the same mechanism (set the field to 0x0000).

```
                +--------+--------+--------+--------+
                | Kind=4 | Len=12 |      Offset     |
                +--------+--------+--------+--------+
                |           Identification          |
                +--------+--------+--------+--------+
                |  Frag. Offset   | Reassy. Checksum|
                +--------+--------+--------+--------+
```

Figure 12   UDP terminal FRAG option format

>> During fragmentation, the UDP header checksum of each fragment needs to be recomputed based on each datagram's pseudoheader.

Unlike the UDP checksum, the reassembly checksum does not need to be updated if the UDP header changes because it covers only the reassembled data. FRAG uses a comparatively weak checksum upon reassembly because the fragments are already checked individually.

>> After reassembly is complete and validated using the checksum of the terminal FRAG option, the UDP header checksum of the resulting datagram needs to be recomputed based on the datagram's pseudoheader.

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 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.

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.

Implementers are advised to limit the space available for UDP reassembly.

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

>> UDP reassembly space limits SHOULD NOT be implemented as an aggregate, 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 additional UDP options, if used, follow the FRAG option in the final fragment and would be included in the reassembled packet. Processing of those options would commence after reassembly. This is especially important for UNSAFE options, which are interpreted only after FRAG.

>> UDP options MUST NOT follow the FRAG header in non-terminal fragments. Any data following the FRAG header in non-terminal fragments MUST be silently dropped. All other options that apply to a reassembled packet MUST follow the FRAG header in the terminal fragment.

In general, UDP packets are fragmented as follows:

1. Create a datagram with data and any non-FRAG UDP options, which we will call "D". Note that the options apply to the entire data area and must follow the data. These options are processed before the rest of the fragmentation steps below.

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 (e.g., OCS).

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 MUST appear in the terminal fragment.

4. For each chunk of "D" in step #3, create a zero-data UDP packet followed by the per-fragment options, with the final option being the FRAG option 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 data as a whole when received.

5. Process the UDP options of each fragment, e.g., computing its OCS.

<<TBD: decide whether it is useful to encode fragments so they can be zero-copied>>

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 packet is processed with any trailing UDP options applying to the reassembled data.

5.6. Maximum Segment Size (MSS)

The Maximum Segment Size (MSS, Kind = 3) option is a 16-bit indicator of the largest UDP segment that can be received. As with the TCP 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].

```
+--------+--------+--------+--------+
| Kind=5 | Len=4  |    MSS size     |
+--------+--------+--------+--------+
```

Figure 13 UDP MSS option format

The UDP MSS option MAY be used 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 to limit the largest reassembled UDP message, 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 set a maximum DPLPMTU.

5.7. Unsafe (UNSAFE)

The Unsafe option (UNSAFE) extends the UDP option space to allow for options that 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 entire UDP payload occurs inside a reassembled set of UDP fragments, such that if UDP fragmentation is not supported, the entire fragment would be silently dropped anyway.

UNSAFE options are an extended option space, with its own additional option types. These are indicated in the first byte after the option Kind as shown in ?, which is followed by the Length. Length is 1 byte for UKinds whose total length (including Kind, UKind, and Length fields) is less than 255 or 2 bytes for larger lengths (in the similar style as the extended option format).

```
                        +--------+--------+--------+
                        | Kind=6 | UKind  | Length |...
                        +--------+--------+--------+
                          1 byte   1 byte  1-2 bytes
```

Figure 14   UDP UNSAFE option format

>> 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 entire reassembled datagram if any fragment or the entire datagram includes an UNSAFE option whose UKind is not supported.

The following UKind values are defined:

UKind	Length	Meaning
0		RESERVED
1-253	(varies)	UNASSIGNED (assignable by IANA)
254	(varies)	RFC 3692-style experiments (UEXP)
255		RESERVED

Experimental UKind EXP ExID values indicate the ExID in the following 2 (or 4) bytes, similar to the UDP EXP option as discussed in Section 5.11. Assigned UDP EXP ExIDs and UDP UNSAFE UKind UEXP ExIDs are assigned from the same registry and can be used either in the EXP option (Section 5.11) or within the UKind UEXP.

5.8. Timestamps (TIME)

The Timestamp (TIME) option exchanges two four-byte 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].

```
        +--------+--------+-------------------+-------------------+
        | Kind=7 | Len=10 |      TSval        |      TSecr        |
        +--------+--------+-------------------+-------------------+
          1 byte   1 byte       4 bytes             4 bytes
```

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 segments 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.

>> TIME SHOULD make this RTT estimate available to the user application.

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 "reply=0" and a reply is defined as both fields 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 [RFC5925].

5.9. Authentication and Encryption (AE)

The Authentication and Encryption (AE) option is intended to allow UDP to provide a similar type of authentication as the TCP Authentication Option (TCP-AO) [RFC5925]. AE the conventional UDP payload and may also cover the surplus area, depending on configuration. It uses the same format as specified for TCP-AO, except that it uses a Kind of 8. AE supports NAT traversal in a similar manner as TCP-AO [RFC6978]. AE can also be extended to provide a similar encryption capability as TCP-AO-ENC, in a similar manner [To18ao].

```
+--------+--------+--------+--------+
| Kind=8 |  Len   |     Digest...   |
+--------+--------+--------+--------+
|          Digest (con't)...        |
+--------+--------+--------+--------+
```

Figure 16   UDP AE option format

Like TCP-AO, AE 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 AE's Key Derivation Function (KDF) uses zeroes as the value for both ISNs. This means that the AE reuses keys when socket pairs are reused, unlike TCP-AO.

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

Like all non-UNSAFE UDP options, AE 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 payload (which is always included), AE can be configured to either include or exclude the surplus area, in a similar way as can TCP-AO can optionally exclude TCP options. When UDP options are covered, the OCS option area checksum and AE hash areas are zeroed before computing the AE 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 option area contents. This is why such dependencies are not permitted except as defined for OCS and UDP-AE.

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

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

The echo request (REQ, kind=9) and echo response (RES, kind=10) options provide a means for UDP options to be used to provide packet-level acknowledgements. Their use is described as part of the

UDP variant of packetization layer path MTU discovery (PLPMTUD) [Fa20]. The options both have the format indicated in Figure 17.

```
               +--------+--------+-----------------+
               |  Kind  | Len=6  |      nonce      |
               +--------+--------+-----------------+
                 1 byte   1 byte       4 bytes
```

Figure 17   UDP REQ and RES options format

5.11. Experimental (EXP)

The Experimental option (EXP) is reserved for experiments [RFC3692]. It uses a Kind value of 254. 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=254 |   Len    |       UDP ExID      |
            +----------+----------+----------+----------+
            |  (option contents, as defined)...         |
            +----------+----------+----------+----------+
```

Figure 18   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]).

Assigned UDP EXP ExIDs and UDP UNSAFE UKind UEXP ExIDs are assigned from the same registry and can be used either in the EXP option or within the UKind UEXP (Section 5.7).

6. 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. For example, FRAG needs to come first if present; new options cannot declare that they need to precede it. The following is a summary of rules for new options and their rationales:

>> New options MUST NOT depend on option space content, excepting only those contained within the UNSAFE option. Only OCS and AE depend on the content of the options themselves and their order is fixed (on transmission, AE is computed first using a zero-checksum

OCS if present, and OCS is computed last before transmission, over the entire option area, including AE).

>> UNSAFE options can both depend on and vary option space 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 contained within the UNSAFE option; areas that need to remain unmodified include the IP header, IP options, the UDP body, the UDP option area (i.e., other options), and the post-option area.

>> 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.

>> New options with fixed lengths smaller than 255 or variable lengths that are always smaller than 255 MUST use only the default option format.

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

o >> FRAG MUST be first, if present, and MUST be processed when encountered (e.g., even before security options).

o >> Only FRAG and UNSAFE options are permitted to modify the UDP body or option areas.

o >> OCS SHOULD be the first option, except in the presence of FRAG, in which case it SHOULD be the first option after FRAG.

7. 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 mandatory options MUST be processed by receivers, if present (presuming UDP options are supported at that receiver).

>> Non-mandatory 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 list.

>> All options except UNSAFE options MUST result in the UDP payload 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 mandatory options.

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

```
        if the UDP checksum fails then
            silently drop (per RFC1122)
        if the UDP checksum passes then
            if OCS is present and fails then
                deliver the UDP payload but ignore all other options
                (this is required to emulate legacy behavior)
            if OCS is present and passes then
                deliver the UDP payload 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.

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

I.e., all options other than OCS 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 other than OCS:

- o if the option is not required by the receiver, then 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 packet.

- o if the packet is accepted (either because the option is not required or because it was required and correct), then pass the option with the 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).

8. 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 receive options that are required. Datagrams not containing these required options MUST be silently dropped and MAY be logged.

- o Extend the receive function to indicate the options and their parameters as received with the corresponding received datagram.

- Extend the send function to indicate the options to be added to the corresponding sent datagram.

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

9. Whose options are these?

UDP options are indicated in an 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 datagrams 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 UDP option area, except as provided by the options selected (e.g., OCS or AE).

10. UDP options FRAG option vs. UDP-Lite

UDP-Lite provides partial checksum coverage, so that 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.

The UDP FRAG options option supports a similar service to UDP-Lite. The main difference is that UDP-Lite provides the un-checksummed user data to the application by default, whereas the UDP FRAG option can safely provide that service only between endpoints that negotiate that capability in advance. An endpoint that does not implement UDP options would silently discard this non-checksummed user data, along with the UDP options as well.

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 separate 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.

11. 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 UDP 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 payload. 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.

(TBD: test with UDP checksum offload and UDP fragmentation offload)

12. 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., ACS).

>> 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 payload fragments as individual transport datagrams.

13. 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 [RFC2140][To20cb]. 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.

[TBD: extend this section to indicate which options MAY vs. MUST NOT be shared and how, e.g., along the lines of To20cb]

14. Updates to RFC 768

This document updates RFC 768 as follows:

- This document defines the meaning of the IP payload area beyond the UDP length but within the IP length.

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

15. 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 surplus area 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 hereby deprecates the requirement asserted in 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 relies on this "matching" of values to avoid needing to transmit both the IP length and UDP length fields, even though this is not a strict requirement of UDP [RFC768] or host requirements [RFC1122] and these preexisting standards were not updated by the ROHC specification. Section A.1.3 of that document is hereby updated to allow for UDP length to vary per packet, so that the UDP length in the table is "CHANGING" rather than "INFERRED". The text that describes the UDP length field this is updated to:

This field is changing as allowed by UDP [RFC768] and used by both UDP options [RFC-TBD] and Teredo extensions [RFC6081] and is therefore classified as CHANGING.

The issue of handling UDP header compression has already been correctly described in more recent specifications, e.g., Sec. 10.10 of Static Context Header Compression [RFC8724]. In that description, the UDP length can be compressed out of a packet only when it can be correctly inferred from the UDP length, i.e., when neither UDP options nor Teredo extensions are present:

"The parser MUST NOT label this field unless the UDP Length value matches the Payload Length value from the IPv6 header."

16. 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.

17. Security Considerations

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 payload 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 Extension option (Section 5.9). 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. Implementations concerned with the potential for this vulnerability MAY implement only the required options and MAY also limit processing of TLVs. 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 three in a row), this limits their DOS impact.

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 packet).

UDP fragmentation introduces its own set of security concerns, which can be handled in a manner similar to IP fragmentation. In particular, the number of 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.

18. 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 5. Additional values in this registry are to be assigned from the UNASSIGNED values in Section 5 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 6.

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 the UDP EXP option or the UDP UNSAFE option when using UKind=UEXP. 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 6.

Upon publication, IANA is hereby requested to create a new registry for UDP UNSAFE UKind numbers. There are no initial assignments in this registry. Values in this registry are to be assigned from the UNASSIGNED values in Section 5.7 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 6.

19. References

19.1. Normative References

[Fa20] Fairhurst, G., T. Jones, "Datagram PLPMTUD for UDP Options," draft-fairhurst-tsvwg-udp-options-dplpmtud, Mar. 2020.

[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.

[RFC3095] Bormann, C. (Ed), et al., "RObust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed," RFC 3095, July 2001.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words," RFC 2119, May 2017.

19.2. Informative References

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

[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.

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

[RFC2140] Touch, J., "TCP Control Block Interdependence," RFC 2140, Apr. 1997.

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

[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.

[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.

[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.

[To18ao] Touch, J., "A TCP Authentication Option Extension for Payload Encryption," draft-touch-tcp-ao-encrypt, Jul. 2018.

[To20cb] Touch, J., M. Welzl, S. Islam, J. You, "TCP Control Block Interdependence," draft-touch-tcpm-2140bis, Apr. 2020.

20. Acknowledgments

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

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 include OCS
net.ipv4.udp_opt_acs	0	Default include ACS
net.ipv4.udp_opt_mss	0	Default include MSS
net.ipv4.udp_opt_time	0	Default include TIME
net.ipv4.udp_opt_frag	0	Default include FRAG
net.ipv4.udp_opt_ae	0	Default include AE

Socket options (sockopt), cached for outgoing datagrams:

Name	meaning
UDP_OPT	Enable UDP options (at all)
UDP_OPT_OCS	Enable UDP OCS option
UDP_OPT_ACS	Enable UDP ACS option
UDP_OPT_MSS	Enable UDP MSS option
UDP_OPT_TIME	Enable UDP TIME option
UDP_OPT_FRAG	Enable UDP FRAG option
UDP_OPT_AE	Enable UDP AE option

Send/sendto parameters:

(TBD - currently using cached 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