TSVWG
Internet Draft
Intended status: Standards Track
Intended updates: 768
Expires: September 2019

J. Touch
Independent consultant

March 5, 2019

Transport Options for UDP
draft-ietf-tsvwg-udp-options-06.txt

Status of this Memo

Copyright Notice

Abstract

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

Table of Contents

1. Introduction....3
2. Conventions used in this document....3
3. Background....3
4. The UDP Option Area....4
5. UDP Options....7
5.1. End of Options List (EOL)....8
5.2. No Operation (NOP)....9
5.3. Option Checksum (OCS)....9
5.4. Alternate Checksum (ACS)....10
5.5. Lite (LITE)....11
5.6. Maximum Segment Size (MSS)....13
5.7. Fragmentation (FRAG)....14
5.8. Coupling FRAG with LITE....16
5.9. Timestamps (TIME)....17
5.10. Authentication and Encryption (AE)....18
6. Echo request (REQ) and echo response (RES)....19
6.1. Experimental (EXP)....19
7. Rules for designing new options....19
8. Option inclusion and processing....20
9. UDP API Extensions....22
10. Whose options are these?....22
11. UDP options LITE option vs. UDP-Lite....23
12. Interactions with Legacy Devices....24
13. Options in a Stateless, Unreliable Transport Protocol....24
14. UDP Option State Caching....25
15. Updates to RFC 768....25
16. Multicast Considerations....25
17. Security Considerations....26
18. IANA Considerations....26
19. References....27
19.1. Normative References....27
19.2. Informative References....27
20. Acknowledgments....29

Appendix A. Implementation Information.....30

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 experimental 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", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lowercase uses of these words are not to be interpreted as carrying significance described in RFC 2119.

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 header and an area for header options. 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 as well. One example of such uses is Substrate Protocol for User Datagrams (SPUD) [Tr16], and 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 experimentally 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 12).

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".

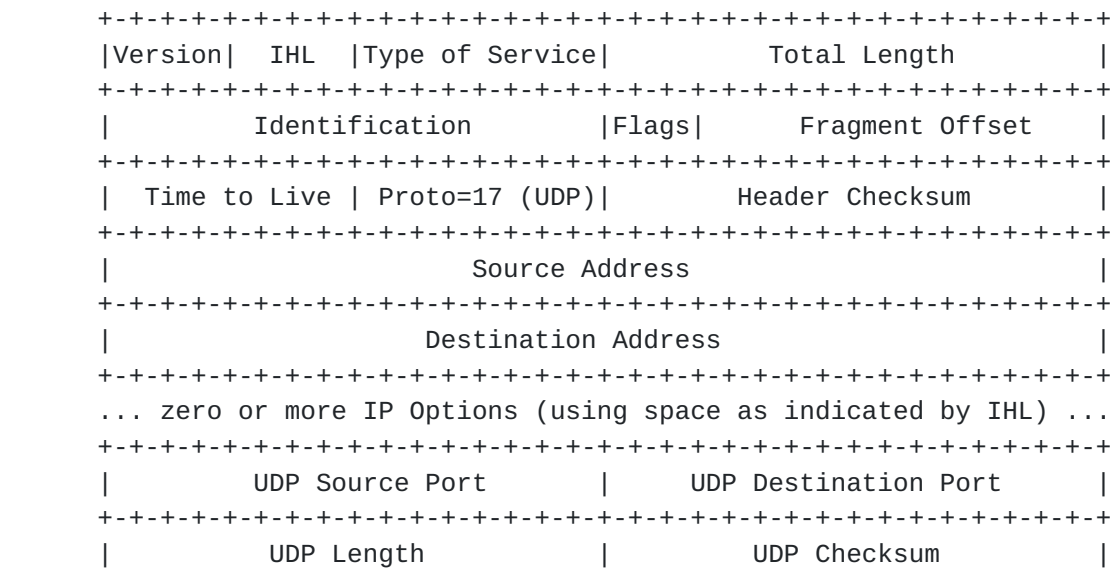

Figure 1 IPv4 datagram with UDP transport payload

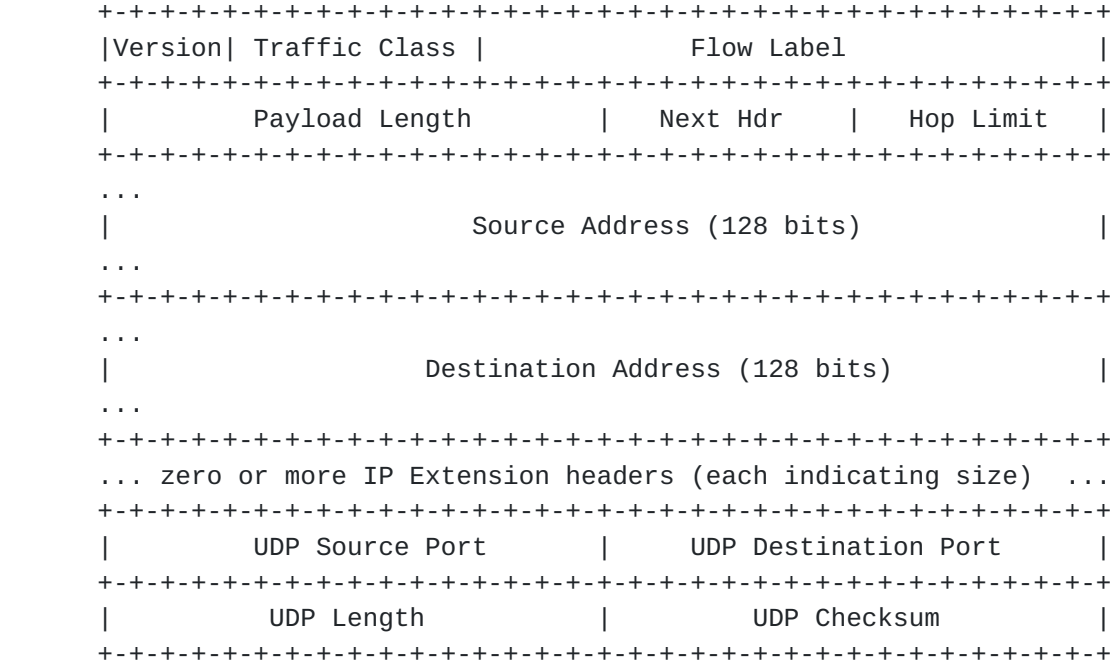

Figure 2 IPv6 datagram with UDP transport payload

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).

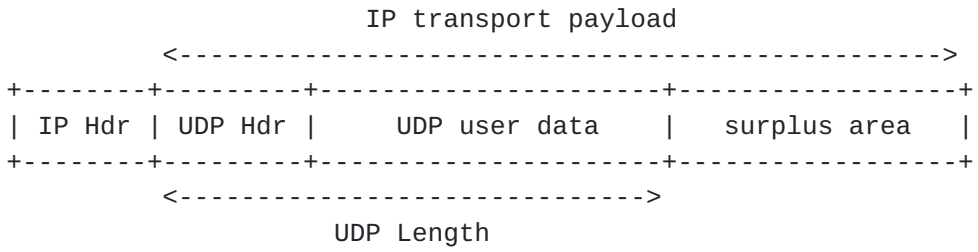

Figure 3 IP transport payload vs. UDP Length

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 12). 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.

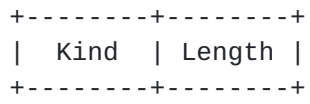

Figure 4 UDP option default format

>> UDP options MAY occur 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. Values outside this range MUST be silently discarded as invalid and logged where rate-limiting permits.

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*	2	Option checksum (OCS)
3*	4	Alternate checksum (ACS)
4*	4	Lite (LITE)
5*	4	Maximum segment size (MSS)
6*	8/10	Fragmentation (FRAG)
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	N(>=4)	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, LITE, FRAG, and MSS. This includes both recognizing and being able to generate these options if configured to do so.

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

>> Receivers MUST silently ignore unknown options. That includes options whose length does not indicate the specified value.

>> Except for NOP, each option SHOULD NOT occur more than once in a single UDP datagram. If a non-NOP option 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).

>> Required options MUST come before other options. Each required option MUST NOT occur more than once (if they are repeated in a received segment, all except the first MUST be silently ignored).

The requirement that required 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.

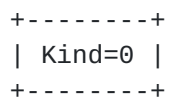

Figure 5 UDP EOL option format

>> When the UDP options do not consume the entire option area, the last non-NOP option SHOULD be EOL (vs. filling the entire option area with NOP values).

>> All bytes after EOL MUST be ignored by UDP option processing. As a result, there can only ever be one EOL option (even if other bytes were zero, they are ignored).

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.

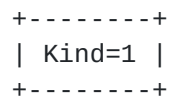

Figure 6 UDP NOP option format

>> 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) is conventional Internet checksum that covers all of the UDP options. The primary purpose of OCS is to detect non-standard (i.e., non-option) uses of the options area.

OCS is calculated by computing the ones-complement of the 8-bit ones-complement checksum (i.e., Internet checksum) sum of the options area. OCS protects the option area from errors in a similar way that the UDP checksum protects the UDP user data (when not zero).

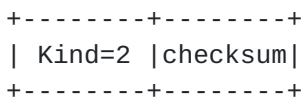

Figure 7 UDP OCS option format

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

>> OCS SHOULD be half-word aligned to the start of the UDP packet. This is to help ensure that the option, together with the other options, result in an overall zero ones-complement sum, which may help the UDP options traverse devices that incorrectly attempt to checksum the surplus area (as originally proposed as the Checksum Compensation Option [Fa18]).

OCS covers the UDP option area, including the Lite option as formatted before swapping (or relocation) for transmission (or, equivalently, after the swap/relocation after reception).

>> If the option checksum fails, all options MUST be ignored and any trailing surplus data (and Lite data, if used) silently discarded.

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

Note that use of the UDP checksum is optional. When not used, the field is zero, where it is also assumed to be "correct" for these purposes.

Note also that OCS is intended to check for accidental errors, not for attacks.

5.4. Alternate Checksum (ACS)

The Alternate Checksum (ACS) provides a stronger alternative to the checksum in the UDP header, using a 16-bit CRC of the conventional UDP payload only (excluding the IP pseudoheader, UDP header, and UDP options, and not include the LITE area). Because it does not include the IP pseudoheader or UDP header, it need not be updated by NATs when IP addresses or UDP ports are rewritten. Its purpose is to detect errors that the UDP checksum, when used, might not detect.

CRC-CCITT (polynomial x^16 + x^12 + x^5 + 1) has been chosen because of its ubiquity and use in other packet protocols, such as X.25, HDLC, and Bluetooth. The option contains FCS-16 as defined in Appendix C of [RFC1662], except that it is not inverted in the final step and that it is stored in the ACS option in network byte order.

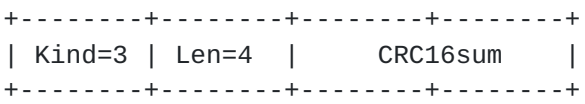

Figure 8 UDP ACS option format

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.

5.5. Lite (LITE)

The Lite option (LITE) is intended to provide equivalent capability to the UDP Lite transport protocol [RFC3828]. UDP Lite allows the UDP checksum to cover only a prefix of the UDP data payload, to protect critical information (e.g., application headers) but allow potentially erroneous data to be passed to the user. This feature helps protect application headers but allows for application data errors. Some applications are impacted more by a lack of data than errors in data, e.g., voice and video.

>> When LITE is active, it MUST come first in the UDP options list.

LITE is intended to support the same API as for UDP Lite to allow applications to send and receive data that has a marker indicating the portion protected by the UDP checksum and the portion not protected by the UDP checksum.

LITE includes a 2-byte offset that indicates the length of the portion of the UDP data that is not covered by the UDP checksum.

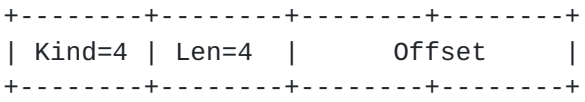

Figure 9 UDP LITE option format

At the sender, the option is formed using the following steps:

1. Create a LITE option, ordered as the first UDP option (Figure 10).

2. Calculate the location of the start of the options as an absolute offset from the start of the UDP header and place that length in the last two bytes of the LITE option.

3. If the LITE data area is 4 bytes or longer, swap all four bytes of the LITE option with the first 4 bytes of the LITE data area (Figure 11). If the LITE data area is 0-3 bytes long, slide the LITE option to the front of the LITE data area (i.e., placing the 0-3 bytes of LITE data after the LITE option).

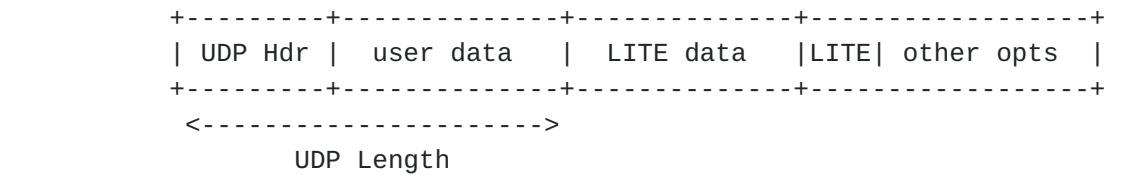

Figure 10 LITE option formation - LITE goes first

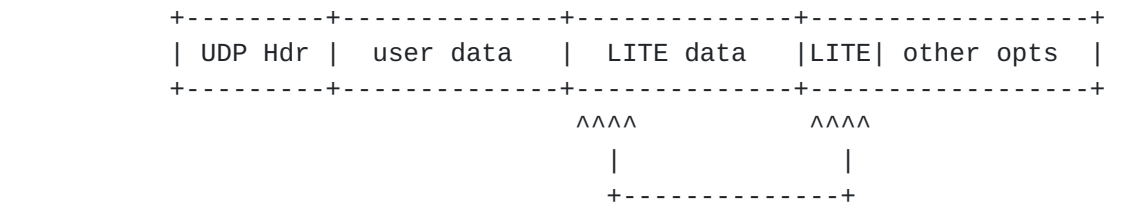

Figure 11 Before sending swap LITE option and front of LITE data

The resulting packet has the format shown in Figure 12. Note that the UDP length now points to the LITE option, and the LITE option points to the start of the option area.

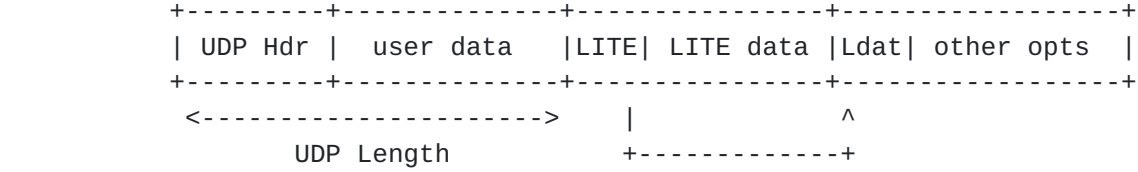

Figure 12 Lite option as sent

A legacy endpoint receiving this packet will discard the LITE option and everything that follows, including the lite data and remainder

of the UDP options. The UDP checksum will protect only the user data, not the LITE option or lite data.

Receiving endpoints capable of processing UDP options will do the following:

1. Process options as usual. This will start at the LITE option.

2. When the LITE option is encountered, record its location as the start of the LITE data area and (if the LITE offset indicates a LITE data length of at least 4 bytes) swap the four bytes there with the four bytes at the location indicated inside the LITE option, which indicates the start of all of the options, including the LITE one (one past the end of the lite data area). If the LITE offset indicates a LITE data area of 0-3 bytes, then slide the LITE option forward that amount and slide the corresponding bytes after the LITE option to where the LITE option originally began. In either case, this restores the format of the option as it was prior to being sent, as per Figure 10.

3. Continue processing the remainder of the options, which are now in the format shown in Figure 11.

The purpose of this swap (or slide) is to support the equivalent of UDP Lite operation together with other UDP options without requiring the entire LITE data area to be moved after the UDP option area.

5.6. Maximum Segment Size (MSS)

The Maximum Segment Size (MSS, Kind = 3) 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].

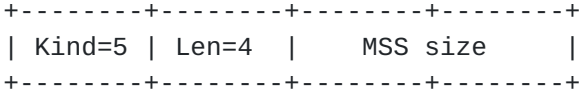

Figure 13 UDP MSS option format

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]).

5.7. Fragmentation (FRAG)

The Fragmentation option (FRAG) 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.

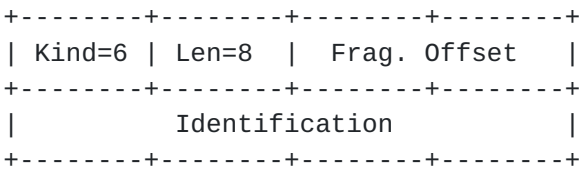

Figure 14 UDP non-terminal FRAG option format

The FRAG option also lacks a "more" bit, zeroed for the terminal fragment of a set. This is possible because the terminal FRAG option is indicated as a longer, 10-byte variant, which 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 15.

>> 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).

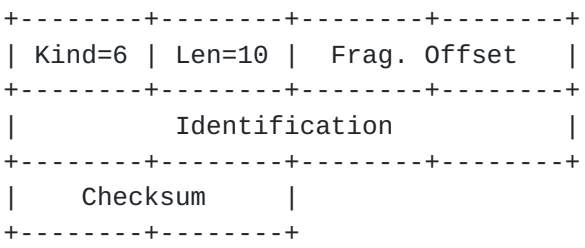

Figure 15 UDP terminal FRAG option format

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

>> 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=8) 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.

FRAG needs to be used with extreme care because it will present incorrect datagram boundaries to a legacy receiver, unless encoded as LITE data (see Section 5.8).

>> A host SHOULD indicate FRAG support by transmitting an unfragmented datagram using the Fragmentation option (e.g., with Offset zero and length 12, i.e., including the checksum area), except when encoded as LITE.

>> A host MUST NOT transmit a UDP fragment before receiving recent confirmation from the remote host, except when FRAG is encoded as LITE.

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 options.

Any additional UDP options would follow the FRAG option in the final fragment, and would be included in the reassembled packet. Processing of those options would commence after reassembly.

>> 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.

5.8. Coupling FRAG with LITE

FRAG can be coupled with LITE to avoid impacting legacy receivers. Each fragment is sent as LITE un-checksummed data, where each UDP packet contains no legacy-compatible data. Legacy receivers interpret these 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 header of such a packet would appear as shown in Figure 16 and Figure 17.

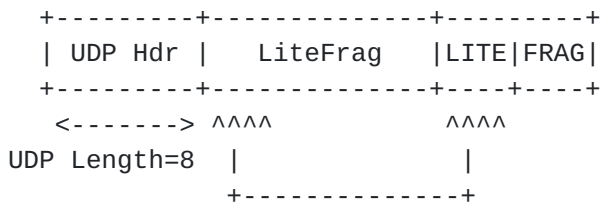

Figure 16 Preparing FRAG as Lite data

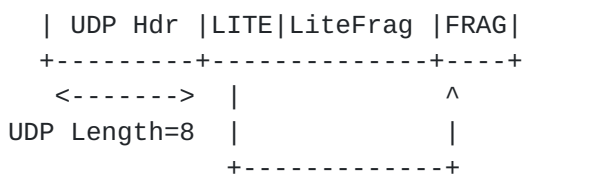

Figure 17 Lite option before transmission

When a packet is reassembled, it appears as a complete LITE data region. The UDP header of the reassembled packet is adjusted accordingly, so that the reassembled region now appears as conventional UDP user data, and processing of the UDP options continues, as with the non-LITE FRAG variant.

5.9. Timestamps (TIME)

The UDP Timestamp option (TIME) 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].

Figure 18 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.

>> UDP 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.

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

These UDP timestamps are modeled after TCP timestamps and have similar expectations. In particular, they are expected to be:

- Values are monotonic and non-decreasing
- Values should increase according to a typical 'tick' time
- A request is defined as "reply=0" and a reply is defined as both fields being non-zero.

- A receiver should always respond to a request with the highest TSval received, which is not necessarily the most recently received.

5.10. Authentication and Encryption (AE)

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

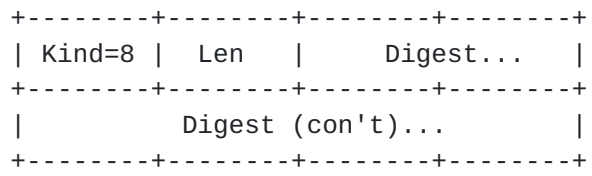

Figure 19 UDP non-terminal FRAG option format

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

UDP-AE can be configured to either include or exclude UDP options, the same way as can TCP-AO. When UDP options are covered, the OCS option area checksum and UDP-AE hash areas are zeroed before computing the UDP-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, UDP-AE can be configured to support NAT traversal, excluding one or both of the UDP ports [RFC6978].

6. 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) [Fa19]. The options both have the format indicated in Figure 20.

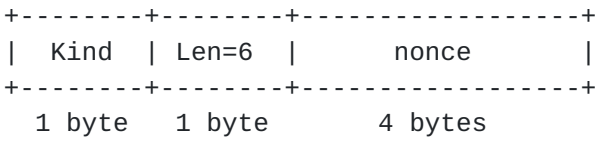

Figure 20 Echo option format

6.1. 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].

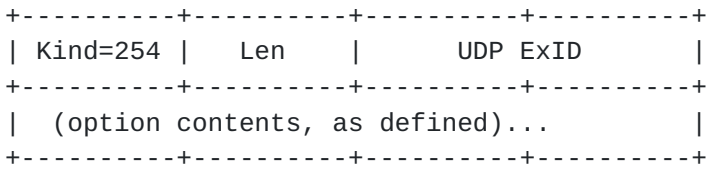

Figure 21 UDP EXP option format

>> 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]).

7. 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, LITE 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. Only OCS, ACS, 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 or ACS if present, and OCS or ACS is computed second over the entire option area, including AE).

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

>> New options MUST NOT modify content anywhere except within their option field; 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.

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

- >> LITE MUST be first, if present, and MUST be processed when encountered (e.g., even before security options).
- >> LITE is the only option that modifies the UDP body or option areas.
- >> OCS or ACS MUST be the first option, except in the presence of LITE, in which case they MUST be the first option after LITE.
- >> OCS and ACS MUST be processed by receivers when encountered in the options list.
- >> AE MUST be processed by receivers when encountered in the options list.

8. 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, based on API settings.

The basic premise is that 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 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

        if both sender and receiver choose to use ACS then
            if ACS passes then
                deliver the UDP payload after parsing
                and processing the rest of the options
            if ACS fails then
                silently drop the packet
            if ACS is not present when received then
                silently drop the packet

        if the sender includes ACS
        and the receiver does not indicate ACS is required then
                the receiver accepts all UDP payloads that pass
                the UDP checksum and indicate for each packet
                whether ACS succeeded, but never drop when ACS fails
```

I.e., ACS should be treated like any other option, 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 (by API configuration) should the receiver require it being present and correct.

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

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

- if the option is required and missing or incorrectly formed, silently drop the packet.

- 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).

9. UDP API Extensions

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

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

This API is extended to support options as follows:

- Extend the method to create receive ports to include receive options that are required. Datagrams not containing these required options MUST be silently dropped and MAY be logged.
- 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.

10. 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. 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, ACS, or AE).

11. UDP options LITE 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 LITE UDP 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 LITE UDP 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.

12. 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)

13. 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.

>> UDP options MUST allow for silent failure on first receipt.

>> UDP options that rely on soft-state exchange MUST allow for message reordering and loss.

>> A UDP option MUST be silently optional until confirmed by exchange with an endpoint.

The above requirements prevent using any option that cannot be safely ignored unless that capability has been negotiated with an endpoint in advance for a socket pair. Legacy systems would need to be able to interpret the transport payload fragments as individual transport datagrams.

14. 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][To19cb]. 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 To19cb]

15. 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.

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 5.10). 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. 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. Note that when a packet's options cannot be processed, it MUST be discarded; the packet and its options should always share the same fate.

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 by IESG Approval or Standards Action [RFC8126].

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]. This registry is initially empty. Values in this registry are to be assigned by IANA using first-come, first-served (FCFS) rules [RFC8126].

19. References

19.1. Normative References

[Fa19] Fairhurst, G., T. Jones, M. Tuexen, I. Ruengeler, T. Voelker, "Packetization Layer Path MTU Discovery for Datagram Transports," draft-ietf-tsvwg-datagram-plpmtud, Feb. 2019.

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

[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.

[RFC1662] Simpson, W. Ed., "PPP in HDLC-like Framing," RFC 1662, Oct. 1994.

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.

[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.

[RFC4960] Stewart, R. (Ed.), "Stream Control Transmission Protocol", RFC 4960, September 2007.

[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.

[RFC5925] Touch, J., A. Mankin, R. Bonica, "The TCP Authentication Option," RFC 5925, June 2010.

[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.

[To19cb] Touch, J., M. Welzl, S. Islam, J. You, "TCP Control Block Interdependence," draft-touch-tcpm-2140bis, Jan. 2019.

[Tr16] Trammel, B. (Ed.), M. Kuelewind (Ed.), "Requirements for the design of a Substrate Protocol for User Datagrams (SPUD)," draft-trammell-spud-req-04, May 2016.

20. Acknowledgments

This work benefitted from feedback from Bob Briscoe, Ken Calvert, Ted Faber, Gorry Fairhurst, C. M. Heard (including the FRAG/LITE combination), 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_lite	0	Default include LITE
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_LITE	Enable UDP LITE 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