TCPM WG
Internet Draft
Updates: 793
Intended status: Standards Track
Expires: April 2022

J. Touch
Independent consultant
Wes Eddy
MTI Systems

October 12, 2021

TCP Extended Data Offset Option
draft-ietf-tcpm-tcp-edo-11.txt

Status of this Memo

Copyright Notice

Abstract

TCP segments include a Data Offset field to indicate space for TCP options but the size of the field can limit the space available for complex options such as SACK and Multipath TCP and can limit the combination of such options supported in a single connection. This document updates RFC 793 with an optional TCP extension to that space to support the use of multiple large options. It also explains why the initial SYN of a connection cannot be extending a single segment.

Table of Contents

1. Introduction..3
2. Conventions used in this document...............................3
3. Motivation..3
4. Requirements for Extending TCP's Data Offset....................4
5. The TCP EDO Option..4
   5.1. EDO Supported..5
   5.2. EDO Extension..5
   5.3. The two EDO Extension variants.............................8
6. TCP EDO Interaction with TCP....................................9
   6.1. TCP User Interface...9
   6.2. TCP States and Transitions.................................9
   6.3. TCP Segment Processing....................................10
   6.4. Impact on TCP Header Size.................................10
   6.5. Connectionless Resets.....................................11
   6.6. ICMP Handling...11
7. Interactions with Middleboxes..................................12
   7.1. Middlebox Coexistence with EDO............................12
   7.2. Middlebox Interference with EDO...........................13
8. Comparison to Previous Proposals...............................14
   8.1. EDO Criteria..14
   8.2. Summary of Approaches.....................................15
   8.3. Extended Segments...16
   8.4. TCPx2...16
   8.5. LO/SLO..17
   8.6. LOIC..17
   8.7. Problems with Extending the Initial SYN...................18
9. Implementation Issues..19
10. Security Considerations.......................................20
11. IANA Considerations...20
12. References..20

12.1. Normative References....................................20
12.2. Informative References..................................20
13. Acknowledgments...22

1. Introduction

TCP's Data Offset (DO)is a 4-bit field, which indicates the number of 32-bit words of the entire TCP header [RFC793]. This limits the current total header size to 60 bytes, of which the basic header occupies 20, leaving 40 bytes for options. These 40 bytes are increasingly becoming a limitation to the development of advanced capabilities, such as when SACK [RFC2018][RFC6675] is combined with either Multipath TCP [RFC6824], TCP-AO [RFC5925], or TCP Fast Open [RFC7413].

This document specifies the TCP Extended Data Offset (EDO) option, and is independent of (and thus compatible with) IPv4 and IPv6. EDO extends the space available for TCP options, except for the initial SYN and SYN/ACK. This document also explains why the option space of the initial SYN segments cannot be extended as individual segments without severe impact on TCP's initial handshake and the SYN/ACK limitation that results from potential middlebox misbehavior. Multiple other TCP extensions are being considered in the TCPM working group in order to address the case of SYN and SYN/ACK segments [Bo14][Br14][To18]. Some of these other extensions can work in conjunction with EDO (e.g., [To18]).

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. Lower case uses of these words are not to be interpreted as carrying RFC-2119 significance.

In this document, the characters ">>" preceding an indented line(s) indicates a compliance requirement statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the explicit compliance requirements of this RFC.

3. Motivation

TCP supports headers with a total length of up to 15 32-bit words, as indicated in the 4-bit Data Offset field [RFC793]. This accounts

for a total of 60 bytes, of which the default TCP header fields occupy 20 bytes, leaving 40 bytes for options.

TCP connections already use this option space for a variety of capabilities. These include Maximum Segment Size (MSS) [RFC793], Window Scale (WS) [RFC7323], Timestamp (TS) [RFC7323], Selective Acknowledgement (SACK) [RFC2018][RFC6675], TCP Authentication Option (TCP-AO) [RFC5925], Multipath TCP (MP-TCP)_[RFC6824], and TCP User Timeout [RFC5482]. Some options occur only in a SYN or SYN/ACK (MSS, WS), and others vary in size when used in SYN vs. non-SYN segments.

Each of these options consumes space, where some options consuming as much space as available (SACK) and other desired combinations can easily exceed the currently available space. For example, it is not currently possible to use TCP-AO with both TS and MP-TCP in the same non-SYN segment, i.e., to combine accurate round-trip estimation, authentication, and multipath support in the same connection - even though these options can be negotiated during a SYN exchange (10 for TS, 16 for TCP-AO, and 12 for MP-TCP).

TCP EDO is intended to overcome this limitation for non-SYN segments, as well as to increase the space available for SACK blocks. Further discussion of the impact of EDO and existing options is discussed in Section 6.4. Extending SYN segments is much more complicated, as discussed in Section 8.7.

4. Requirements for Extending TCP's Data Offset

The primary goal of extending the TCP Data Offset field is to increase the space available for TCP options in all segments except the initial SYN.

An important requirement of any such extension is that it not impact legacy endpoints. Endpoints seeking to use this new option should not incur additional delay or segment exchanges to connect to either new endpoints supporting this option or legacy endpoints without this option. We call this a "backward downgrade" capability.

An additional consideration of this extension is avoiding user data corruption in the presence of popular network devices, including middleboxes. Consideration of middlebox misbehavior can also interfere with extension in the SYN/ACK.

5. The TCP EDO Option

TCP EDO extends the option space for all segments except the initial SYN (i.e., SYN set and ACK not set) and SYN/ACK response. EDO is

indicated by the TCP option codepoint of EDO-OPT and has two types: EDO Supported and EDO Extension, as discussed in the following subsections.

5.1. EDO Supported

EDO capability is determined in both directions using a single exchange of the EDO Supported option (Figure 1). When EDO is desired on a given connection, the SYN and SYN/ACK segments include the EDO Supported option, which consists of the two required TCP option fields: Kind and Length. The EDO Supported option is used only in the SYN and SYN/ACK segments and only to confirm support for EDO in subsequent segments.

```
                         +--------+--------+
                         |  Kind  | Length |
                         +--------+--------+
```

Figure 1 TCP EDO Supported option

An endpoint seeking to enable EDO includes the EDO Supported option in the initial SYN. If receiver of that SYN agrees to use EDO, it responds with the EDO Supported option in the SYN/ACK. The EDO Supported option does not extend the TCP option space.

>> Connections using EDO MUST negotiate its availability during the SYN exchange of the initial three-way handshake.

>> An endpoint confirming and agreeing to EDO use MUST respond with the EDO Supported option in its SYN/ACK.

The SYN/ACK uses only the EDO Supported option (and not the EDO Extension option, below) because it may not yet be safe to extend the option space in the reverse direction due to potential middlebox misbehavior (see Section 7.2). Extension of the SYN and SYN/ACK space is addressed as a separate option (see Section 8.7).

5.2. EDO Extension

When EDO is successfully negotiated, all other segments use the EDO Extension option, of which there are two variants (Figure 2 and Figure 3). Both variants are considered equivalent and either variant can be used in any segment where the EDO Extension option is required. Both variants add a Header_Length field (in network-standard byte order), indicating the length of the entire TCP header in 32-bit words. Figure 3 depicts the longer variant, which includes an additional Segment_Length field, which is identical to the TCP

pseudoheader TCP Length field and used to detect when segments have been altered in ways that would interfere with EDO (discussed further in Section 5.3).

```
                +--------+--------+--------+--------+
                |  Kind  | Length |  Header_Length  |
                +--------+--------+--------+--------+
```

Figure 2 TCP EDO Extension option - simple variant

```
                +--------+--------+--------+--------+
                |  Kind  | Length |  Header_Length  |
                +--------+--------+--------+--------+
                |  Segment_Length |
                +--------+--------+
```

Figure 3 TCP EDO Extension option - with segment length verification

>> Once enabled on a connection, all segments in both directions MUST include the EDO Extension option. Segments not needing extension MUST set the EDO Extension option Header Length field equal to the Data Offset length.

>> The EDO Extension option MAY be used only if confirmed when the connection transitions to the ESTABLISHED state, e.g., a client is enabled after receiving the EDO Supported option in the SYN/ACK and the server is enabled after seeing the EDO Extension option in the final ACK of the three-way handshake. If either of those segments lacks the appropriate EDO option, the connection MUST NOT use any EDO options on any other segments.

Internet paths may vary after connection establishment, introducing misbehaving middleboxes (see Section 7.2). Using EDO on all segments in both directions allows this condition to be detected.

>> The EDO Supported option MAY occur in an initial SYN as desired (e.g., as expressed by the user/application) and in the SYN/ACK as confirmation, but MUST NOT be inserted in other segments. If the EDO Supported option is received in other segments, it MUST be silently ignored.

>> If EDO has not been negotiated and agreed, the EDO Extension option MUST be silently ignored on subsequent segments. The EDO Extension option MUST NOT be sent in an initial SYN segment or SYN/ACK, and MUST be silently ignored and not acknowledged if so received.

>> If EDO has been negotiated, any subsequent segments arriving without the EDO Extension option MUST be silently ignored. Such events MAY be logged as warning errors and logging MUST be rate limited.

When processing a segment, EDO needs to be visible within the area indicated by the Data Offset field, so that processing can use the EDO Header_length to override the field for that segment.

>> The EDO Extension option MUST occur within the space indicated by the TCP Data Offset.

>> The EDO Extension option indicates the total length of the header. The EDO Header_length field MUST NOT exceed that of the total segment size (i.e., TCP Length).

>> The EDO Header Length MUST be at least as large as the TCP Data Offset field of the segment in which they both appear. When the EDO Header Length equals the Data Offset length, the EDO Extension option is present but it does not extend the option space. When the EDO Header Length is invalid, the TCP segment MUST be silently dropped.

>> The EDO Supported option SHOULD be aligned on a 16-bit boundary and the EDO Extension option SHOULD be aligned on a 32-bit boundary, in both cases for simpler processing.

For example, a segment with only EDO would have a Data Offset of 6 or 7 (depending on the EDO Extension variant used), where EDO would be the first option processed, at which point the EDO Extension option would override the Data Offset and processing would continue until the end of the TCP header as indicated by the EDO Header_length field.

There are cases where it might be useful to process other options before EDO, notably those that determine whether the TCP header is valid, such as authentication, encryption, or alternate checksums. In those cases, the EDO Extension option is preferably the first option after a validation option, and the payload after the Data Offset is treated as user data for the purposes of validation.

>> The EDO Extension option SHOULD occur as early as possible, either first or just after any authentication or encryption, and SHOULD be the last option covered by the Data Offset value.

Other options are generally handled in the same manner as when the EDO option is not active, unless they interact with other options.

One such example is TCP-AO [RFC5925], which optionally ignores the contents of TCP options, so it would need to be aware of EDO to operate correctly when options are excluded from the HMAC calculation.

>> Options that depend on other options, such as TCP-AO [RFC5925] (which may include or exclude options in MAC calculations) MUST also be augmented to interpret the EDO Extension option to operate correctly.

5.3. The two EDO Extension variants

There are two variants of the EDO Extension option; one includes a copy of the TCP segment length, copied from the TCP pseudoheader [RFC793]. The Segment_Length field is added to the longer variant to detect when segments are incorrectly and inappropriately merged by middleboxes or TCP offload processing but without consideration for the additional option space indicated by the EDO Header_Length field. Such effects are described in further detail in Section 7.2.

>> An endpoint MAY use either variant of the EDO Extension option interchangeably.

When the longer, 6-byte variant is used, the Segment_Length field is used to check whether modification of the segment was performed consistent with knowledge of the EDO option. The Segment_Length field will detect any modification of the length of the segment, such as might occur when segments are split or merged, that occurs without also updating the Segment Length field as well. The Segment Length field thus helps endpoints detects devices that merge or split TCP segments without support for EDO. Devices that merge or split TCP segments that support EDO would update the Segment Length field as needed but would also ensure that the user data is handled separately from the extended option space indicate by EDO.

>> When an endpoint creates a new segment using the 6-byte EDO Extension option, the Segment_Length field is initialized with a copy of the segment length from the TCP pseudoheader.

>> When an endpoint receives a segment using the 6-byte EDO Extension option, it MUST validate the Segment_Length field with the length of the segment as indicated in the TCP pseudoheader. If the segment lengths do not match, the segment MUST be discarded and an error SHOULD be logged in a rate-limited manner.

>> The 6-byte EDO Extension variant SHOULD be used where middlebox or TCP offload support could merge or split TCP segments without

consideration for the EDO option. Because these conditions could occur at either endpoint or along the network path, the 6-byte variant SHOULD be preferred until sufficient evidence for safe use of the 4-byte variant is determined by the community.

The field will not detect other modification of the TCP user data; such modifications would need more complex detection mechanisms, such as checksums or hashes. When these are used, as with IPsec or TCP-AO, the 4-byte variant is sufficient.

>> The 4-byte EDO Extension variant is sufficient when EDO is used in conjunction with other mechanisms that provide integrity protection, such as IPsec or TCP-AO.

6. TCP EDO Interaction with TCP

The following subsections describe how EDO interacts with the TCP specification [RFC793].

6.1. TCP User Interface

The TCP EDO option is enabled on a connection using a mechanism similar to any other per-connection option. In Unix systems, this is typically performed using the 'setsockopt' system call.

>> Implementations can also employ system-wide defaults, however systems SHOULD NOT activate this extension by default to avoid interfering with legacy applications.

>> Due to the potential impacts of legacy middleboxes (discussed in Section 7), a TCP implementation supporting EDO SHOULD log any events within an EDO connection when options that are malformed or show other evidence of tampering arrive. An operating system MAY choose to cache the list of destination endpoints where this has occurred with and block use of EDO on future connections to those endpoints, but this cache MUST be accessible to users/applications on the host. Note that such endpoint assumptions can vary in the presence of load balancers where server implementations vary behind such balancers.

6.2. TCP States and Transitions

TCP EDO does not alter the existing TCP state or state transition mechanisms.

6.3. TCP Segment Processing

TCP EDO alters segment processing during the TCP option processing step. Once detected, the TCP EDO Extension option overrides the TCP Data Offset field for all subsequent option processing. Option processing continues at the next option (if present) after the EDO Extension option.

6.4. Impact on TCP Header Size

The TCP EDO Supported option increases SYN header length by a minimum of 2 bytes but could increase it by more depending on 32-bit word alignment. Currently popular SYN options total 19 bytes, which leaves more than enough room for the EDO Supported option:

- o SACK permitted (2 bytes in SYN, optionally 2 + 8N bytes after) [RFC2018][RFC6675]

- o Timestamp (10 bytes) [RFC7323]

- o Window scale (3 bytes) [RFC7323]

- o MSS option (4 bytes) [RFC793]

Adding the EDO Supported option would result in a total of 21 bytes of SYN option space.

Subsequent segments would use 10 bytes of option space without any SACK blocks (TS only; WS and MSS are used only in SYN and SYN/ACK) or allow up to 3 SACK blocks before needing to use EDO; with EDO, the number of SACK blocks or additional options would be substantially increased. There are also other options that are emerging in the SYN, including TCP Fast Open, which uses another 6-18 (typically 10) bytes in the SYN/ACK of the first connection and in the SYN of subsequent connections [RFC7413].

TCP EDO can also be negotiated in SYNs with either of the following large options:

- o TCP-AO (authentication) (16 bytes) [RFC5925]

- o Multipath TCP (12 bytes in SYN and SYN/ACK, 20 after) [RFC6824]

Including TCP-AO with TS, WS, SACK increases the SYN option space use to 35 bytes; with Multipath TCP the use is 31 bytes. When Multipath TCP is enabled with the typical options, later segments would require 30 bytes without SACK, thus limiting the SACK option

to one block unless EDO is also supported on at least non-SYN segments.

The full combination of the above options (47 bytes for TS, WS, MSS, SACK, TCP-AO, and MPTCP) does not fit in the existing SYN option space and (as noted) that space cannot be extended within a single SYN segment. There has been a proposal to change TS to a 2 byte "TS permitted" signal in the initial SYN, provided it can be safely enabled during the connection later or might be avoided completely [Ni15]. Even using "TS-permitted", the total space is still too large to support in the initial SYN without SYN option space extension [Bo14][Br14][To18].

The EDO Extension option has negligible impact on other headers because it can either come first or just after security information, and in either case the additional 4 or 6 bytes are easily accommodated within the TCP Data Offset length. Once the EDO option is processed, the entirety of the remainder of the TCP segment is available for any remaining options.

6.5. Connectionless Resets

A RST may arrive during a currently active connection or may be needed to cleanup old state from an abandoned connection. The latter occurs when a new SYN is sent to an endpoint with matching existing connection state, at which point that endpoint responds with a RST and both ends remove stale information.

The EDO Extension option is mandatory on all TCP segments once negotiated, i.e., except in the SYN and SYN/ACK (which establish support) and the RST. A RST may lack the context to know that EDO is active on a connection.

>> The EDO Extension option MAY occur in a RST when the endpoint has connection state that has negotiated EDO. However, unless the RST is generated by an incoming segment that includes an EDO Extension option, the transmitted RST MUST NOT include the EDO Extension option.

6.6. ICMP Handling

ICMP responses are intended to include the IP and the port fields of TCP and UDP headers of typical TCP/IP and UDP/IP packets [RFC792]. This includes the first 8 data bytes of the original datagram, intended to include the transport port numbers used for connection demultiplexing. Later specifications encourage returning as much of the original payload as possible [RFC1812]. In either case, legacy

options or new options in the EDO extension area might or might not be included, and so options are generally not assumed to be part of ICMP processing anyway.

7. Interactions with Middleboxes

Middleboxes are on-path devices that typically examine or modify packets in ways that Internet routers do not [RFC3234]. This includes parsing transport headers and/or rewriting transport segments in ways that may affect EDO.

There are several cases to consider:

- Typical NAT/NAPT devices, which modify only IP address and/or TCP port number fields (with associated TCP checksum updates)

- Middleboxes that try to reconstitute TCP data streams, such as for deep-packet inspection for virus scanning

- Middleboxes that modify known TCP header fields

- Middleboxes that rewrite TCP segments

7.1. Middlebox Coexistence with EDO

Middleboxes can coexist with EDO when they either support EDO or when they ignore its impact on segment structure.

NATs and NAPTs, which rewrite IP address and/or transport port fields, are the most common form of middlebox and are not affected by the EDO option.

Middleboxes that support EDO would be those that correctly parse the EDO option. Such boxes can reconstitute the TCP data stream correctly or can modify header fields and/or rewrite segments without impact to EDO.

Conventional TCP proxies terminate the TCP connection in both directions and thus operate as TCP endpoints, such as when a client-middlebox and middlebox-server each have separate TCP connections. They would support EDO by following the host requirements herein on both connections. The use of EDO on one connection is independent of its use on the other in this case.

7.2. Middlebox Interference with EDO

Middleboxes that do not support EDO cannot coexist with its use when they modify segment boundaries or do not forward unknown (e.g., the EDO) options.

So-called "transparent" rewriting proxies, which inappropriately and incorrectly modify TCP segment boundaries, might mix option information with user data if they did not support EDO. Such devices might also interfere with other TCP options such as TCP-AO. There are three types of such boxes:

- Those that process received options and transmit sent options separately, i.e., although they rewrite segments, they behave as TCP endpoints in both directions.

- Those that split segments, taking a received segment and emitting two or more segments with revised headers.

- Those that join segments, receiving multiple segments and emitting a single segment whose data is the concatenation of the components.

In all three cases, EDO is either treated as independent on different sides of such boxes or not. If independent, EDO would either be correctly terminated in either or both directions or disabled due to lack of SYN/ACK confirmation in either or both directions. Problems would occur only when TCP segments with EDO are combined or split while ignoring the EDO option. In the split case, the key concern is if the split happens within the option extension space or if EDO is silently copied to both segments without copying the corresponding extended option space contents. However, the most comprehensive study of these cases indicates that "although middleboxes do split and coalesce segments, none did so while passing unknown options" [Ho11].

Note that the second and third types of middlebox behaviors listed above may create syndromes similar to TCP transmit and receive hardware offload engines that incorrectly modify segments with unknown options.

Middleboxes that silently remove options that they do not implement have been observed [Ho11]. Such boxes interfere with the use of the EDO Extension option in the SYN and SYN/ACK segments because extended option space would be misinterpreted as user data if the EDO Extension option were removed, and this cannot be avoided. This is one reason that SYN and SYN/ACK extension requires alternate

mechanisms (see Section 8.7). It is also the reason for the 6-byte EDO Extension variant (see Section 5.3), which can detect such merging or splitting of segments. Further, if such middleboxes become present on a path they could cause similar misinterpretation on segments exchanged in the ESTABLISHED and subsequent states. As a result, this document requires that the EDO Extension option be avoided on the SYN/ACK and that this option needs to be used on all segments once successfully negotiated and encourages use of the 6-byte EDO Extension variant.

Deep-packet inspection systems that inspect TCP segment payloads or attempt to reconstitute the data stream would incorrectly include option data in the reconstituted user data stream, which might interfere with their operation.

>> It can be important to detect misbehavior that could cause EDO space to be misinterpreted as user data. In such cases, EDO SHOULD be used in conjunction with an integrity protection mechanism. This includes the 6-byte EDO Extension variant or stronger mechanisms such as IPsec, TCP-AO, etc. It is useful to note that such protection only helps non-compliant components and enable avoidance (e.g., disabling EDO), but integrity protection alone cannot correct the misinterpretation of EDO space as user data.

This situation is similar to that of ECN and ICMP support in the Internet. In both cases, endpoints have evolved mechanisms for detecting and robustly operating around "black holes". Very similar algorithms are expected to be applicable for EDO.

8. Comparison to Previous Proposals

EDO is the latest in a long line of attempts to increase TCP option space [Al06][Ed08][Ko04][Ra12][Yo11]. The following is a comparison of these approaches to EDO, based partly on a previous summary [Ra12]. This comparison differs from that summary by using a different set of success criteria.

8.1. EDO Criteria

Our criteria for a successful solution are as follows:

o Zero-cost fallback to legacy endpoints.

o Minimal impact on middlebox compatibility.

o No additional side-effects.

Zero-cost fallback requires that upgraded hosts incur no penalty for attempting to use EDO. This disqualifies dual-stack approaches, because the client might have to delay connection establishment to wait for the preferred connection mode to complete. Note that the impact of legacy endpoints that silently reflect unknown options are not considered, as they are already non-compliant with existing TCP requirements [RFC793].

Minimal impact on middlebox compatibility requires that EDO works through simple NAT and NAPT boxes, which modify IP addresses and ports and recompute IPv4 header and TCP segment checksums. Middleboxes that reject unknown options or that process segments in detail without regard for unknown options are not considered; they process segments as if they were an endpoint but do so in ways that are not compliant with existing TCP requirements (e.g., they should have rejected the initial SYN because of its unknown options rather than silently relaying it).

EDO also attempts to avoid creating side-effects, such as might happen if options were split across multiple TCP segments (which could arrive out of order or be lost) or across different TCP connections (which could fail to share fate through firewalls or NAT/NAPTs).

These requirements are similar to those noted in [Ra12], but EDO groups cases of segment modification beyond address and port - such as rewriting, segment drop, sequence number modification, and option stripping - as already in violation of existing TCP requirements regarding unknown options, and so we do not consider their impact on this new option.

8.2. Summary of Approaches

There are three basic ways in which TCP option space extension has been attempted:

1. Use of a TCP option.

2. Redefinition of the existing TCP header fields.

3. Use of option space in multiple TCP segments (split across multiple segments).

A TCP option is the most direct way to extend the option space and is the basis of EDO. This approach cannot extend the option space of the initial SYN.

Redefining existing TCP header fields can be used to either contain additional options or as a pointer indicating alternate ways to interpret the segment payload. All such redefinitions make it difficult to achieve zero-impact backward compatibility, both with legacy endpoints and middleboxes.

Splitting option space across separate segments can create unintended side-effects, such as increased delay to deal with path latency or loss differences.

The following discusses three of the most notable past attempts to extend the TCP option space: Extended Segments, TCPx2, LO/SLO, and LOIC. [Ra12] suggests a few other approaches, including use of TCP option cookies, reuse/overload of other TCP fields (e.g., the URG pointer), or compressing TCP options. None of these is compatible with legacy endpoints or middleboxes.

8.3. Extended Segments

TCP Extended Segments redefined the meaning of currently unused values of the Data Offset (DO) field [Ko04]. TCP defines DO as indicating the length of the TCP header, including options, in 32-bit words. The default TCP header with no options is 5 such words, so the minimum currently valid DO value is 5 (meaning 40 bytes of option space). This document defines interpretations of values 0-4: DO=0 means 48 bytes of option space, DO=1 means 64, DO=2 means 128, DO=3 means 256, and DO=4 means unlimited (e.g., the entire payload is option space). This variant negotiates the use of this capability by using one of these invalid DO values in the initial SYN.

Use of this variant is not backward-compatible with legacy TCP implementations, whether at the desired endpoint or on middleboxes. The variant also defines a way to initiate the feature on the passive side, e.g., using an invalid DO during the SYN/ACK when the initial SYN had a valid DO. This capability allows either side to initiate use of the feature but is also not backward compatible.

8.4. TCPx2

TCPx2 redefines legacy TCP headers by basically doubling all TCP header fields [Al06]. It relies on a new transport protocol number to indicate its use, defeating backward compatibility with all existing TCP capabilities, including firewalls, NATs/NAPTs, and legacy endpoints and applications.

8.5. LO/SLO

The TCP Long Option (LO, [Ed08]) is very similar to EDO, except that presence of LO results in ignoring the existing Data Offset (DO) field and that LO is required to be the first option. EDO considers the need for other fields to be first and declares that the EDO is the last option as indicated by the DO field value. Like LO, EDO is required in every segment once negotiated.

The TCP Long Option draft also specified the SYN Long Option (SLO) [Ed08]. If SLO is used in the initial SYN and successfully negotiated, it is used in each subsequent segment until all of the initial SYN options are transmitted.

LO is backward compatible, as is SLO; in both cases, endpoints not supporting the option would not respond with the option, and in both cases the initial SYN is not itself extended.

SLO does modify the three-way handshake because the connection isn't considered completely established until the first data byte is acknowledged. Legacy TCP can establish a connection even in the absence of data. SLO also changes the semantics of the SYN/ACK; for legacy TCP, this completes the active side connection establishment, where in SLO an additional data ACK is required. A connection whose initial SYN options have been confirmed in the SYN/ACK might still fail upon receipt of additional options sent in later SLO segments. This case - of late negotiation fail - is not addressed in the specification.

8.6. LOIC

TCP Long Options by Invalid Checksum is a dual-stack approach that uses two initial SYNS to initiate all updated connections [Yo11]. One SYN negotiates the new option and the other SYN payload contains only the entire options. The negotiation SYN is compliant with existing procedures, but the option SYN has a deliberately incorrect TCP checksum (decremented by 2). A legacy endpoint would discard the segment with the incorrect checksum and respond to the negotiation SYN without the LO option.

Use of the option SYN and its incorrect checksum both interfere with other legacy components. Segments with incorrect checksums will be silently dropped by most middleboxes, including NATs/NAPTs. Use of two SYNs creates side-effects that can delay connections to upgraded endpoints, notably when the option SYN is lost or the SYNs arrive out of order. Finally, by not allowing other options in the negotiation SYN, all connections to legacy endpoints either use no

options or require a separate connection attempt (either concurrent or subsequent).

8.7. Problems with Extending the Initial SYN

The key difficulty with most previous proposals is the desire to extend the option space in all TCP segments, including the initial SYN, i.e., SYN with no ACK, typically the first segment of a connection, as well as possibly the SYN/ACK. It has proven difficult to extend space within the segment of the initial SYN in the absence of prior negotiation while maintaining current TCP three-way handshake properties, and it may be similarly challenging to extend the SYN/ACK (depending on asymmetric middlebox assumptions).

A new TCP option cannot extend the Data Offset of a single TCP initial SYN segment and cannot extend a SYN/ACK in a single segment when considering misbehaving middleboxes. All TCP segments, including the initial SYN and SYN/ACK, may include user data in the payload data [RFC793], and this can be useful for some proposed features such as TCP Fast Open [RFC7413]. Legacy endpoints that ignore the new option would process the payload contents as user data and send an ACK. Once ACK'd, this data cannot be removed from the user stream.

The Reserved TCP header bits cannot be redefined easily, even though three of the six total bits have already been redefined (ECE/CWR [RFC3168] and NS [RFC3540]). Legacy endpoints have been known to reflect received values in these fields; this was safely dealt with for ECN but would be difficult here [RFC3168].

TCP initial SYN (SYN and not ACK) segments can use every other TCP header field except the Acknowledgement number, which is not used because the ACK field is not set. In all other segments, all fields except the three remaining Reserved header bits are actively used. The total amount of available header fields, in either case, is insufficient to be useful in extending the option space.

The representation of TCP options can be optimized to minimize the space needed. In such cases, multiple Kind and Length fields are combined, so that a new Kind would indicate a specific combination of options, whose order is fixed and whose length is indicated by one Length field. Most TCP options use fields whose size is much larger than the required Kind and Length components, so the resulting efficiency is typically insufficient for additional options.

The option space of an initial SYN segment might be extended by using multiple initial segments (e.g., multiple SYNs or a SYN and non-SYN) or based on the context of previous or parallel connections. This method may also be needed to extend space in the SYN/ACK in the presence of misbehaving middleboxes. Because of their potential complexity, these approaches are addressed in separate documents [Bo14][Br14][To18].

Option space cannot be extended in outer layer headers, e.g., IPv4 or IPv6. These layers typically try to avoid extensions altogether, to simplify forwarding processing at routers. Introducing new shim layers to accommodate additional option space would interfere with deep-packet inspection mechanisms that are in widespread use.

As a result, EDO does not attempt to extend the space available for options in TCP initial SYNs. It does extend that space in all other segments (including SYN/ACK), which has always been trivially possible once an option is defined.

9. Implementation Issues

TCP segment processing can involve accessing nonlinear data structures, such as chains of buffers. Such chains are often designed so that the maximum default TCP header (60 bytes) fits in the first buffer. Extending the TCP header across multiple buffers may necessitate buffer traversal functions that span boundaries between buffers. Such traversal can also have a significant performance impact, which is additional rationale for using TCP option space - even extended option space - sparingly.

Although EDO can be large enough to consume the entire segment, it is important to leave space for data so that the TCP connection can make forward progress. It would be wise to limit EDO to consuming no more than MSS-4 bytes of the IP segment, preferably even less (e.g., MSS-128 bytes).

When using the ExID variant for testing and experimentation, either TCP option codepoint (253, 254) is valid in sent or received segments.

Implementers need to be careful about the potential for offload support interfering with this option. The EDO data needs to be passed to the protocol stack as part of the option space, not integrated with the user segment, to allow the offload to independently determine user data segment boundaries and combine them correctly with the extended option data. Some legacy hardware receive offload engines may present challenges in this regard, and

may be incompatible with EDO where they incorrectly attempt to process segments with unknown options. Such offload engines are part of the protocol stack and updated accordingly. Issues with incorrect resegmentation by an offload engine can be detected in the same way as middlebox tampering.

10. Security Considerations

It is meaningless to have the Data Offset further exceed the position of the EDO data offset option.

>> When the EDO Extension option is present, the EDO Extension option SHOULD be the last non-null option covered by the TCP Data Offset, because it would be the last option affected by Data Offset.

This also makes it more difficult to use the Data Offset field as a covert channel.

11. IANA Considerations

We request that, upon publication, this option be assigned a TCP Option codepoint by IANA, which the RFC Editor will replace EDO-OPT in this document with codepoint value.

The TCP Experimental ID (ExID) with a 16-bit value of 0x0ED0 (in network standard byte order) has been assigned for use during testing and preliminary experiments.

12. References

12.1. Normative References

[RFC793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

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

12.2. Informative References

[Al06] Allman, M., "TCPx2: Don't Fence Me In", draft-allman-tcpx2-hack-00 (work in progress), May 2006.

[Bo14] Borman, D., "TCP Four-Way Handshake", draft-borman-tcp4way-00 (work in progress), October 2014.

[Br14] Briscoe, B., "Inner Space for TCP Options", draft-briscoe-tcpm-inner-space-01 (work in progress), October 2014.

[Ed08] Eddy, W. and A. Langley, "Extending the Space Available for TCP Options", draft-eddy-tcp-loo-04 (work in progress), July 2008.

[Ho11] Honda, M., Nishida, Y., Raiciu, C., Greenhalgh, A., Handley, M., and H. Tokuda, "Is it still possible to extend TCP", Proc. ACM Sigcomm Internet Measurement Conference (IMC), 2011, pp. 181-194.

[Ko04] Kohler, E., "Extended Option Space for TCP", draft-kohler-tcpm-extopt-00 (work in progress), September 2004.

[Ni15] Nishida, Y., "A-PAWS: Alternative Approach for PAWS", draft-nishida-tcpm-apaws-02 (work in progress), Oct. 2015.

[Ra12] Ramaiah, A., "TCP option space extension", draft-ananth-tcpm-tcpoptext-00 (work in progress), March 2012.

[RFC792] Postel, J., "Internet Control Message Protocol", RFC 792, September 1981.

[RFC1812] Baker, F. (Ed.), "Requirements for IP Version 4 Routers," RFC 1812, June 1995.

[RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

[RFC3234] Carpenter, B. and S. Brim, "Middleboxes: Taxonomy and Issues", RFC 3234, February 2002.

[RFC3540] Spring, N., Wetherall, D., and D. Ely, "Robust Explicit Congestion Notification (ECN) Signaling with Nonces", RFC 3540, June 2003.

[RFC5482] Eggert, L., and F. Gont, "TCP User Timeout Option", RFC 5482, March 2009.

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

[RFC6675] Blanton, E., Allman, M., Wang, L., Jarvinen, I., Kojo, M., and Y. Nishida, "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP", RFC 6675, August 2012.

[RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, January 2013.

[RFC7323] Borman, D., Braden, B., Jacobson, V., and R. Scheffenegger (Ed.), "TCP Extensions for High Performance", RFC 7323, September 2014.

[RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, December 2014.

[To21] Touch, J., T. Faber, "TCP SYN Extended Option Space Using an Out-of-Band Segment", draft-touch-tcpm-tcp-syn-ext-opt (work in progress), Oct. 2019.

[Yo11] Yourtchenko, A., "Introducing TCP Long Options by Invalid Checksum", draft-yourtchenko-tcp-loic-00 (work in progress), April 2011.

13. Acknowledgments

The authors would like to thank the IETF TCPM WG for their feedback, in particular: Oliver Bonaventure, Bob Briscoe, Ted Faber, John Leslie, Pasi Sarolahti, Richard Scheffenegger, and Alexander Zimmerman.

This work is 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

Wesley M. Eddy
MTI Systems
US

Email: wes@mti-systems.com