TCPM WG
Internet Draft
Updates: 793
Intended status: Standards Track
Expires: January 2015

J. Touch
USC/ISI
Wes Eddy
MTI Systems

July 2, 2014

TCP Extended Data Offset Option
draft-touch-tcpm-tcp-edo-03.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 that have evolved. This document updates RFC 793 with an optional TCP extension to that space to support the use of multiple large options such as SACK with either TCP Multipath or TCP AO. It also explains why the initial SYN of a connection cannot be extending as a single segment.

Table of Contents

1. Introduction....2
2. Conventions used in this document....3
3. Requirements for Extending TCP's Data Offset....3
4. The TCP EDO Option....3
5. TCP EDO Interaction with TCP....6
5.1. TCP User Interface....6
5.2. TCP States and Transitions....6
5.3. TCP Segment Processing....6
5.4. Impact on TCP Header Size....6
5.5. Connectionless Resets....7
5.6. ICMP Handling....8
6. Interactions with Middleboxes....8
7. Comparison to Previous Proposals....9
7.1. EDO Criteria....9
7.2. Summary of Approaches....10
7.3. Extended Segments....11
7.4. TCPx2....11
7.5. LOO/SLO....12
7.6. LOIC....12
7.7. Problems with Extending the Initial SYN....13
8. Security Considerations....14
9. IANA Considerations....14
10. References....14
10.1. Normative References....14
10.2. Informative References....15
11. Acknowledgments....16

1. Introduction

TCP's Data Offset 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] or TCP-AO [RFC5925].

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 segment (i.e., SYN and not ACK, the first segment of a new connection). This document also explains why the option space of a single initial SYN segment cannot be extended without severe impact on TCP's initial handshake.

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. 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.

4. 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). The EDO option is organized as indicated in Figure 1 and Figure 2. For initial SYN segments (i.e., those whose ACK bit is not set), the EDO request option consists of the required Kind and Length fields only. All other segments optionally use the EDO length option, which adds a Header_Length field (in network-standard byte rder), indicating the length of the

entire TCP header in bytes. The codepoint value of the EDO Kind is EDO-OPT.

Figure 1 TCP EDO request option

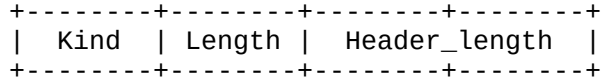

Figure 2 TCP EDO length option

EDO support is determined in both directions using the same exchange. An endpoint seeking to enable EDO support includes the EDO request option in the initial SYN.

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

>> An endpoint confirming EDO support MUST respond with EDO length option in its SYN-ACK.

The EDO length option is required in SYN-ACKs when confirming support for EDO. The SYN-ACK thus can take advantage of EDO, using it to extend its option space. If such extension is not required, then EDO would be equal to 4 * Data Offset (i.e., EDO would indicate in bytes the same length indicated by Data Offset in words).

>> Once negotiated on a connection, EDO MAY be present as needed on other segments in either direction. The EDO option SHOULD NOT be used if the total option space needed can be accommodated by the existing Data Offset field.

>> The EDO request option (i.e., whose option length is 2) MAY occur in an initial SYN as desired (e.g., by the user/application), but MUST NOT be inserted in other segments. If the EDO request option is received, it MUST be silently ignored.

>> The EDO length option MAY occur in segments other than the initial SYN if EDO has been negotiated on a connection. If EDO has not been negotiated and agreed, the EDO length option MUST be silently ignored. The EDO length option MUST NOT be sent in an initial SYN segment, and MUST be silently ignored and not acknowledged if so received.

EDO extends the space available for options, but does not consume space unless needed. Segments that don't need the additional space can use the existing Data Offset field as currently specified [RFC793]. 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 Data Offset for that segment.

>> The EDO length option MUST occur within the length of the TCP Data Offset.

>> The EDO length 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 SHOULD be a multiple of 4 to simplify processing.

>> The EDO request option SHOULD be aligned on a 16-bit boundary and the EDO length 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, where EDO would be the first option processed, at which point the EDO length 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 length 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 length 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 length option to operate correctly.

5. TCP EDO Interaction with TCP

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

5.1. 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' function.

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

5.2. TCP States and Transitions

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

5.3. TCP Segment Processing

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

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.

5.4. Impact on TCP Header Size

The TCP EDO request option increases SYN header length by a minimum of 2 bytes. Currently popular SYN options total 19 bytes, which leaves more than enough room for the EDO request:

- SACK permitted (2 bytes in SYN, optionally 2 + 8N bytes after) [RFC2018][RFC6675]
- Timestamp (10 bytes) [RFC1323]

- Window scale (3 bytes) [RFC1323]

- MSS option (4 bytes) [RFC793]

Adding the EDO option would result in a total of 21 bytes of SYN option space. Subsequent segments would use 19 bytes of option space without any SACK blocks or allow up to 4 SACK blocks before needing to use EDO; with EDO, the number of SACK blocks or additional options would be substantially increased.

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

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

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

Including TCP-AO increases the SYN option space use to 37 bytes; with Multipath TCP the use is 33 bytes. When Multipath TCP is enabled with the typical options, later segments might require 39 bytes without SACK, thus effectively disabling the SACK option unless EDO is also supported on at least non-SYN segments.

The full combination of the above options (49 bytes including EDO) 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 [Ni14]. Even using "TS-permitted", the total space is still too large to support in the initial SYN without SYN option space extension [To14].

The EDO 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 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.

5.5. 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 option is not mandatory in any TCP segment, except the SYN and SYN-ACK of the three-way handshake to establish its support.

>> The EDO length 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 option, the RST MUST NOT include the EDO length option.

5.6. 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.

6. Interactions with Middleboxes

Any new TCP option may be impacted by the presence of any on-path device that examines or modifies transport headers [RFC3234]. Boxes that parse or modify TCP options need to follow the same requirements of TCP endpoints in supporting EDO, or they could interfere with connections. The primary concern is so-called "transparent" rewriting proxies, which modify TCP segment boundaries and thus would mix option information with user data if they do not support EDO. Such devices interfere with many other TCP options, and although their use is not common they would interfere with connections that use EDO.

More common are NATs, which rewrite IP address and/or transport port fields. NATs are not affected by the EDO option.

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

It may 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 integrity protection mechanisms, such as IPsec, TCP-AO, etc. It is useful to note that such protection helps find only non-compliant components.

?? should we include a header checksum, as suggested by Oliver Bonaventure, too? Should it be integrated with EDO or independent? If so, we either need one of the following:

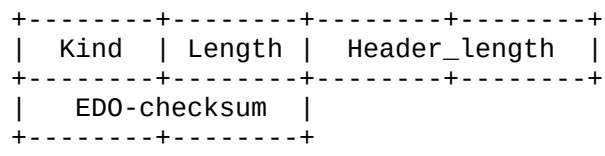

Figure 3 TCP EDO length + sum option

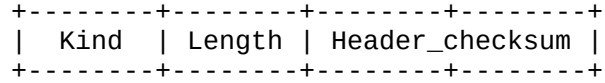

Figure 4 TCP header checksum option

The former adds a header checksum to EDO - this could be optional or required, and its presence can be determined from the option length. The checksum would be over only the additional EDO space.

The latter requires a new option and we'd need to decide what the option covered. It could cover just the option space.

Finally, another option would be to try to make rewriting middleboxes fail by some other, more opaque means - such as using DO=0 when EDO is used as suggested by John Leslie (but this would also require considering whether EDO would be required on all packets, or just on DO=0, as well as where EDO would be and whether it could precede security/integrity options such as TCP-AO).

7. 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.

7.1. EDO Criteria

Our criteria for a successful solution are as follows:

- Zero-cost fallback to legacy endpoints.
- Minimal impact on middlebox compatibility.

- 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.

7.2. 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, LOO/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.

7.3. 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.

7.4. 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.

7.5. LOO/SLO

The TCP Long Option (LO, [Ed08]) is very similar to EDO, except that presence of LO results in ignoring the existing 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. Unlike LO, EDO is not required in every segment once negotiated, saving 6 bytes if not actively needed.

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 ACKed. 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.

7.6. 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).

7.7. 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. It has proven difficult to extend space in this initial SYN in the absence of prior negotiation while maintaining current TCP three-way handshake properties.

A new TCP option cannot extend the Data Offset of a single TCP initial SYN segment. All TCP segments, including the initial SYN, may include user data in the payload data [RFC793], and this can be useful for some proposed features such as TCP Fast Open [Ch14]. 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 six Reserved TCP header bits cannot be redefined easily, because the original specification did not require their contents to be ignored, even though three of those bits have already been redefined (ECE/CWR [RFC3168] and NS [RFC3540]). Legacy endpoints are required to drop TCP segments where those bits are not zero, making it difficult to assume backward downgrade.

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. Because of their potential complexity, these approaches are addressed in a separate document [To14].

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.

8. Security Considerations

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

>> When the EDO length option is present, the EDO length 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.

9. 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.

This section is to be removed prior to publication as an RFC.

10. References

10.1. Normative References

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

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

10.2. Informative References

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

[Ch14] Cheng, Y., Chu, J., and A. Jain, "TCP Fast Open", draft-ietf-tcpm-fastopen-09, June 2014.

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

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

[Ni14] Nishida, Y., "A-PAWS: Alternative Approach for PAWS", draft-nishida-tcpm-apaws-01 (work in progress), June 2014.

[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.

[RFC1323] Jacobson, V., Braden, R., and D. Borman, "TCP Extensions for High Performance", RFC 1323, May 1992.

[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.

[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.

[To14] Touch, J., (et al., TBD), "Extending TCP Option Space During Connection Establishment", draft-touch-tcp-syn-edo-00 (work in progress), July 2014.

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

11. Acknowledgments

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

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

Authors' Addresses

Joe Touch
USC/ISI
4676 Admiralty Way
Marina del Rey, CA 90292-6695 USA

Phone: +1 (310) 448-9151
Email: touch@isi.edu

Wesley M. Eddy
MTI Systems
US

Email: wes@mti-systems.com