Implementation of the TCP Extended Data Offset Option

Harry Trieu, Joe Touch, Ted Faber
USC/ISI
4676 Admiralty Way, Suite 1001
Marina del Rey, CA 90292
310-822-1511
htrieu@usc.edu, touch@isi.edu, faber@isi.edu

ABSTRACT

TCP Extended Data Offset (EDO) Option extends the space available for TCP options. The current TCP specification allows a maximum of 40 bytes of options to be sent with each segment. Certain use cases require a combination of options that exceed 40 bytes in total size, motivating the need to extend the space available for options. This project explores the implementation of TCP extended data offset option in the Linux 3.13 kernel and testing of the implementation.

1. INTRODUCTION

TCP is a popular protocol that facilitates reliable transmission of data between network devices. TCP is extended by options that customize its behavior [5].

Software engineers have expressed interest in boosting the responsiveness of their network applications through the use of multiple, lengthy TCP options. The current specification of TCP accounts for options using a fixed length field that is too small for many modern applications. As a result, developers must compromise and choose a subset of options that fit the fixed length field. This complicates tasks that rely on the use of options such as tuning the network performance of applications.

TCP EDO removes this limitation by extending the fixed length field while maintaining backwards compatibility with the legacy protocol. Furthermore, the option will increase the customizability of TCP for current and future network application development. The implementation of this option demonstrates that the proposed specification can be reasonably implemented in a modern operating system and serve as a reference for implementation in other operating systems.

During the process of implementing this option, the implementers developed a test harness to validate the functionality and discovered inherent characteristics of Linux that made implementation challenging.

This paper explores the current specification of TCP options and its limits, the proposed Extended Data Offset (EDO) option, a plan for implementation, issues encountered during implementation, and the results of completed work.

2. BACKGROUND

The original TCP specification was written in 1981, and TCP continues to be the dominant Internet protocol in use today [6]. TCP options give TCP the ability to evolve with changes in technology. Without options, the evolution of TCP would come to a halt and network innovation would have to take place at a different layer of the protocol stack.

TCP is being deployed in new network environments with increased security and performance requirements. New options are being developed to facilitate and take advantage of these deployments. The original TCP specification reserves a finite amount of space for options and this space is hindering the development of complex options. Increasing option space is necessary to ensure the long-term viability of TCP and the Internet.

A number of options have been introduced to make TCP viable in the modern world. The Window Scale Option allows TCP to function in networks with higher capacity than the original specification intended [2]. The Authentication and MD5 Options allows TCP to function in environments where the level of security provided by the original specification is inadequate [1].

Increasing option space is a relatively straightforward task, however increasing option space while maintaining backwards compatibility with legacy implementations of TCP is challenging. Network devices running legacy implementations of TCP are common on the Internet. Because all options reside in the data section of a TCP segment, a legacy implementation might interpret the options in the EDO extension area as data and pass the options up to a user application. This could present problems ranging from a poorly rendered webpage to the introduction of security vulnerabilities.

Non-experimental TCP options consist of one to three fields. The Option-Kind field (1-byte long) is required and serves as a unique identifier for the option. The Option-Length field (1-byte long) indicates the length of the entire option, including the Option-Kind and Option-Length fields, in bytes. The Option-Data field contains data relevant to the option (if applicable) [5].

Experimental TCP options introduce an Experiment-Identifier (ExID) field (2 or 4-bytes long), which distinguishes experiments from one another. The ExID field immediately follows the Option-Length field. The Option-Kind field of an Experimental TCP option contains the experimental codepoints 253 or 254 [8]. This implementation properly uses experimental TCP option formatting.

TCP options are stored in the data portion of a TCP segment, reducing the amount of space available for the payload. The standard TCP header contains a 4-bit Data Offset field. Data Offset indicates the number of 32-bit words in the TCP header or the offset into the segment where user data lives. The Data Offset field can have a maximum value of 15, allowing it to account for TCP headers up to 60 bytes in size. Because the standard TCP header occupies 20 bytes, a maximum of 40 bytes are available to options.

The Data Offset field becomes a limitation when a combination of TCP options greater than 40 bytes in size needs to be used. This

motivates the need to develop a method to increase the space available for options.

3. OVERVIEW OF EDO

TCP Extended Data Offset (EDO) option extends the space available to TCP options using a Header Length field that overrides the Data Offset field. When EDO is used, the legacy TCP Data Offset field indicates the size of the standard 20-byte TCP header, checksum options, and the EDO option. EDO is the last option covered by Data Offset [9].

There are two variants of the EDO option – EDO-REQUEST and EDO-LENGTH. EDO-REQUEST is sent with the initial SYN segment to negotiate bidirectional support for EDO. It is 2 bytes in length. The endpoint initiating the connection will include the EDO-REQUEST option in the SYN segment. If the remote endpoint supports EDO, it will respond with a SYN-ACK segment containing a null EDO-LENGTH option.

When the endpoint that initiated the connection receives the SYN-ACK containing a null EDO-LENGTH option, it may begin using EDO. The endpoint that sent the SYN-ACK may begin using EDO when it receives an ACK from the connection initiator containing a null or non-null EDO-LENGTH option.

The EDO-LENGTH option is 4 bytes in length and contains a Header Length field, which indicates the size of the entire TCP header, including extended options, in 4-byte words. A null EDO-LENGTH option has a Header Length field, which contains the same value as the legacy Data Offset field.

EDO must maintain backwards compatibility with legacy endpoints. This makes extending the option space in the initial SYN difficult, as bidirectional support for EDO has yet to be negotiated. The use of EDO in SYN segments may be explored in the future [10].

4. PHASES AND TESTBED

TCP EDO was implemented in the Linux 3.13 kernel. It was necessary to divide the implementation into the following five phases to make testing of each phase possible. (1) Build support for sending of a TCP option called JUNK in order to explicitly surpass the 40-byte limit for options. (2) Modify iperf, a TCP/UDP bandwidth measurement tool, to support the use of EDO and JUNK options. Iperf was used to test the implementation. (3) Modify TCP send routines to enable sending of the EDO option. (4) Modify TCP receive routines to support receiving and processing of EDO and options covered by EDO. (5) Modify TCP handshaking code to support negotiation of EDO.

Compilation of the modified kernel and testing of EDO took place on DETERLab,[4] a cyber-security experimentation facility. Two DETERLab machines directly connected using a 1 Gb/s link were used for testing. Wireshark, a modified version of iperf3, and a simple client/server application were used for testing.

4.1 The Junk Option

Before implementing EDO, it was necessary to develop a TCP option, coined "JUNK", that could explicitly exceed 40 bytes in length. This JUNK option consists of the standard Option-Kind=253 (Experimental) and Option-Length fields, an ExID field (0xF81B, for testing and not registered), and a payload containing test characters.

The JUNK option is enabled on a per-connection basis using the setsockopt system call. The length of the JUNK payload is governed by the proc file /proc/sys/net/ipv4/tcp_junk_length. If the JUNK option is enabled, JUNK is sent in all segments in which available option space permits.

4.2 iperf3 Modification

iperf3 was used to test the performance and functionality of EDO. Two command line flags were added: –E which enables EDO on the iperf connection, and –K which enables JUNK on the iperf connection.

4.3 EDO Send

Implementing sending of the EDO-LENGTH option was similar to implementing sending of the JUNK option, with a few exceptions. EDO-LENGTH contains a Header Length field that represents the number of 4-byte words in the entire TCP header, so Header Length needed to be calculated and written to this field instead of the standard Data Offset field.

EDO sending can be enabled or disabled as a system-wide default by using the proc file system to write to the file /proc/sys/net/ipv4/tcp_edo. Setting EDO enabled as default is not advised, as it interferes with connections to non-EDO capable hosts. A socket option was also implemented to enable EDO on a per-connection basis.

4.4 EDO Receive

The kernel was modified to recognize and process TCP segments containing the EDO-LENGTH option. By default, the routines that copy data into userspace and acknowledge segments do not take the new header length into account. This causes options to be copied into userspace and acknowledgements to acknowledge both options and data.

Both issues were resolved by storing the new header length in the TCP per-segment control block soon after it becomes known and updating routines that refer to the old Data Offset value such that they refer to the control block member variable.

4.5 EDO Negotiation

If a machine has /proc/sys/net/ipv4/tcp_edo set to 1, it will respond to segments containing the EDO-REQUEST option with a segment containing a null EDO-LENGTH option, confirming support for EDO. The machine will not send any segments that actually utilize EDO once the connection is established, however it will process incoming segments containing the EDO-LENGTH option.

If a machine is initiating the connection and EDO is enabled on that connection and tcp_edo is enabled on that machine, the EDO-REQUEST option will be included with the SYN. If the machine is not initiating the connection, it will respond to segments containing the EDO-REQUEST option with a segment containing the null EDO-LENGTH option, confirming its support for EDO. Regardless of whether the machine initiates a connection or receives a connection request containing an EDO-REQUEST option, the machine will send segments that utilize EDO on an established connection when necessary.

5. RESULTS AND ISSUES

5.1 Basic Results

The initial tests validated that EDO-enabled hosts participate in EDO negotiation when requested on a per-connection basis, and that EDO-disabled hosts do not.

Throughput tests were performed using the JUNK option with a length of 8 bytes. Generic Receive Offload (GRO), a kernel

feature that merges packets with nearly identical headers to reduce the overhead of processing many small packets, was disabled before running the test [3]. A throughput of 932 Mb/sec was observed when EDO was enabled. By comparison, throughput with EDO disabled, JUNK disabled, and GRO enabled was 940 Mb/sec. Throughput did not decrease significantly when EDO was used.

Tests confirmed the maximum size of options that could fit in a TCP segment with EDO enabled. The largest byte-aligned JUNK payload was 248 bytes, as limited by the option length field, ExID, and need to reserve room for the option kind and length. A TCP segment containing 272 bytes of options yielded a throughput of 773 Mbits/sec. The 272 byte overall total limit is imposed by the amount of room allocated for the TCP header in Linux (a constant called MAX_TCP_HEADER).

5.2 Issues Encountered

Leaving GRO enabled caused throughput to drop from 932 Mb/sec to 208 Mb/sec. The number of data retransmissions also grew from 0 to over 6000. Several tests were performed in an attempt to pinpoint the cause of GRO's behavior. Packets must have identical TCP timestamps in order to be considered candidates for merging [3]. GRO does not know how to interpret EDO so merging two packets containing the EDO-LENGTH option will result in corrupt data. We attempted to trick GRO into deeming two consecutive EDO packets ineligible for merging by flipping bit 15 in the EDO header length field with each outgoing segment on the connection. This did not cause the subpar throughput and retransmissions to go away, leading us to believe that GRO merges consecutive packets even if their EDO-LENGTH option differed in value, which is inconsistent with correct GRO operation. Additional tests varying socket buffer sizes resulted in no retransmissions with buffers under 5191, and significant retransmissions above that, with a peak at 16K. The number of retransmissions received roughly corresponded to the number of delayed selective acknowledgements sent.

Because throughput with EDO enabled and GRO disabled did not deviate significantly from throughput with EDO disabled and GRO enabled, it was not clear whether GRO was effectively reducing CPU utilization. CPU usage on the receiving machine during 5-minute iperf test runs indicated negligible differences in CPU load. The testbed machines may be fast enough that GRO makes a negligible impact on CPU utilization.

Generic implementation issues were also uncovered in the process of implementing EDO. If an endpoint changes its maximum segment size after EDO has been negotiated, this can lead to problems. In addition, it is not clear how Linux deals with a connection that has more options enabled than can fit (with EDO on or off). The total size of options is calculated when a TCP segment is being constructed, rather than when options are set using the setsockopt system call. An error handling mechanism may need to be implemented to deal with this problem.

5.3 Debugging

Debugging the EDO implementation required a cumbersome combination of print statements, writing to and reading from the proc filesystem, capturing packets using Wireshark, and observing the output of netstat. Because Wireshark operates at the link layer and does not know how to interpret EDO, it often considered legitimate segments to be TCP retransmissions [5]. This is due to the fact that Wireshark expects the sequence number of an incoming segment to equal the sequence number plus length of data in the last segment received. Wireshark interprets options accounted for by EDO as data, thus it expects a larger next sequence number.

6. CONCLUSION

Implementation of TCP EDO is feasible in a modern, widely used operating system. The space available for options in a TCP segment was increased from the standard 40 bytes to 272 bytes (this includes the 8-byte EDO-LENGTH option). The 272 byte limit is imposed by the amount of headroom allocated for the TCP header in the Linux implementation.

Using EDO with the Generic Receive Offload (GRO) engine enabled results in dropped packets, but the reason remains unconfirmed. As a result, GRO should be disabled when using EDO until an EDO-compatible version of GRO is developed. Using EDO on a TCP connection results in a negligible performance impact, making EDO feasible for use in high-performance network environments.

In the future, this work may explore the 272-byte option space limit and development of an EDO-compatible GRO engine.

7. ACKNOWLEDGMENTS

We would like to thank the DETER Project for providing the computing resources necessary to implement TCP EDO. We would also like to thank the USC Office of the Provost for funding this project.

8. REFERENCES

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

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

[3] Corbet, J. (2009, October 27). JLS2009: Generic receive offload. Retrieved December 7, 2014, from https://lwn.net/Articles/358910/

[4] Mirkovic, J., Benzel, T., Faber, T., Braden R., Wroclawski, J. and Schwab, S., "The DETER Project: Advancing the Science of Cyber Security Experimentation and Test," in *Proceedings of the IEEE HST '10 Conference*, Waltham MA, November 2010.

[5] Postel, J., "Transmission Control Protocol," RFC 793, September 1981.

[6] Reynders, D., and Wright, E. (2003). *Practical TCP/IP and Ethernet Networking for Industry*. Burlington: Elsevier.

[7] Risso, F., and Degioanni, L., "An architecture for high performance network analysis," *Computers and Communications, 2001. Proceedings. Sixth IEEE Symposium on* , vol., no., pp.686,693, 2001
doi: 10.1109/ISCC.2001.935450

[8] Touch, J., "Shared Use of Experimental TCP Options," RFC 6994, August 2013.

[9] Touch, J., and Eddy, W., "TCP Extended Data Offset Option," draft-ietf-tcpm-tcp-edo-01, October 2014.

[10] Touch, J., and Faber, T., "TCP SYN Extended Option Space Using an Out-of-Band Segment", draft-touch-tcpm-tcp-syn-ext-opt-01 (work in progress), September 2014.