Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: November 27, 2010

A. Biswas
NetApp, Inc.

May 26, 2010

Support for Stronger Error Detection Codes in TCP for Jumbo Frames
draft-anumita-tcpm-stronger-checksum-00

Abstract

There is a class of data serving protocols and applications that cannot tolerate undetected data corruption on the wire. Data corruption could occur at the source in software, in the network interface card, out on the link, on intermediate routers or at the destination network interface card or node. The Ethernet CRC and the 16-bit checksum in the TCP/UDP headers are used to detect data errors. Most applications rely on these checksums to detect data corruptions and do not use any checksums or CRC checks at their level. Research has shown that the TCP/UDP checksums are catching a significant number of errors, however, the research suggests that one packet in 10 billion will have an error that goes undetected for Ethernet MTU frames (MTU of 1500). Under certain situations, "bad" hosts can introduce undetected errors at a much higher frequency and order. With the use of Jumbo frames on the rise, and therefore more data bits on the wire that could be corrupted, the current 16-bit TCP/UDP checksum, or the Ethernet 32-bit CRC are simply not sufficient for detecting errors. This document specifies a proposal to use stronger checksum algorithms for TCP Jumbo Frames for IPv4 and IPv6 networks. The Castagnoli CRC 32C algorithm used in iSCSI and SCTP is proposed as the error detection code of choice.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on November 27, 2010.

Table of Contents

1. Introduction

There is a class of data serving applications that host business and financial data. Detecting and recovering from data corruption is paramount to the success of this class of applications. Data corruption can occur while data is transiting from the source to a desired destination. Data can get corrupted right at the source due to software errors, within the network interface card, out on the wire or link, in intermediate routers and at the destination network interface or node. Link errors are detected using the Ethernet 32-bit CRC. Node or router errors are detected using the 16-bit checksum in the transport headers of TCP and UDP. Most applications do not have built-in error detection capability and typically rely on the checksums in the underlying networking layers. Stone et al. [Stone] have recommended applications employ their own checksums to detect errors that go undetected by lower levels. They have made this recommendation for the standard Ethernet MTU. They have done so considering situations where a "bad" host can introduce undetected errors at a much higher frequency and order. It must also be said that the physical layer already does encodings with bit error rates(BER) of 10^-12 to 10^-14 and therefore the current checksum algorithms may be sufficient. However, stronger checksumming accounts for the cases where noisy hardware, bad cables can introduce noise at a much higher frequency and order. It is also to be noted that increasing speed of the physical medium (to 40G and 100G) can also lead to higher BER.

Another dynamic, very much in the rise is the use and deployment of Jumbo Frames. Jumbo Frames reduce per packet overheads significantly and are a cheap way of improving the performance of bulk data applications. Combining the use of Jumbo frames with noisy physical medium increases the risk of undetected bit errors as there simply are more bits that can get corrupted. This is rather concerning as business and financial data typically are transported over the network using file access based protocols like NFS, CIFS, HTTP over TCP.

The strength of the Ethernet CRC checksum and the 16-bit Transport checksum has been found to reduce for data segments that are larger than the standard Ethernet MTU. Koopman et. al. [Koopman] have explored a number of CRC polynomials as well as the polynomial used in the Ethernet CRC calculation. They have measured the effectiveness of these CRC polynomials for different data word lengths, where a data word is a bit stream from 64 bits to 128 Kbits. These data word lengths cover lengths equivalent to Ethernet MTUs and Jumbo frames and also frame lengths larger than Jumbo frames. They found that the Castagnoli polynomial x^32 + x^28 + x^27 + x^26 + x^25 + x^23 + x^22 + x^20 + x^19 + x^18 + x^14 + x^13 + x^11 + x^10 + x^9

+ x^8 + x^6 + x^0 represented as the 32-bit code 0x8F6E37A0 bests other CRC polynomials for Jumbo frames and larger segments. This polynomial has been adopted by the iSCSI and SCTP standards. It is to be noted that this polynomial is represented as the 32-bit code 0x11EDC6F41 in SCTP in accordance to the convention adopted for bit-ordering at the transport-level, i.e., bit-ordering for mapping SCTP messages to polynomials is that bytes are taken most significant first, but within each bytes, bits are taken least-significant first.

Given the ubiquity of TCP, it is the layer where we can introduce stronger error detection capability without duplicating the effort in higher layers. TCP options provide an easy path to introduce stronger checksum without hindering interoperability. TCP options allow a TCP stack supporting a TCP option to interoperate seamlessly with a TCP stack that does not support the new TCP option (RFC 1122 [RFC1122] requires the interoperability in Section 4.2.2.5).

This document proposes that the use of the Castagnoli polynomial, also known as the CRC 32C as the "checksum" of choice for TCP protocol. Other summation based checksum algorithms like Fletcher and Adler's algorithm were evaluated in RFC 3385 [RFC3385] and found to behave substanially worse than CRCs and hence are not considered in this proposal.

By standardizing a stronger checksum at the TCP level, we can quickly drive the offloading of this checksum to NIC hardware, just as the 16-bit TCP checksum is offloaded by most NIC vendors today. Offloading computation to hardware allows us to get rid of the in-software computation overheads of stronger checksum algorithms.

Another positive effect of implementing strong TCP checksumming is that this will drive the rapid adoption of 9K Jumbo frames and make it considerably easier to consider even larger Jumbo Frames.

1.1. Conventions

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].

2. Calculating the CRC-32C value

The 16-bit TCP checksum does a checksum of the TCP header and payload. It also includes the pseudo header values of Source Address, Destination Address, Protocol and TCP Length. The addition of the bytes of a pseudo header into a summation based checksum algorithm is simpler than the inclusion of the bytes of a pseudo

header into a CRC computation. This is because a CRC computation assumes a contiguous bit stream when translating the bit stream to a polynomial for doing the polynomial division. The psuedo header was added to the TCP checksum computation in order to detect errors introduced in one of the IP header fields that could possibly cause the packet to be sent to an incorrect destination. These fields also get included in the IP header checksum. The intent was to include them in two separate checksums for better data integrity. One can question the need for including the pseudo-header fields twice. The pseudo-header currently get included thrice if one considers the fact that the Ethernet CRC is computed over the entire Ethernet frame and Ethernet is ubiquitous today. So for the purposes of this draft, all the fields used in the current TCP checksum except the pseudo-header must be included in the CRC-32c calculation. If this draft's proposal is accepted for standardization, IETF may elect to add back the pseudo-header into the CRC-32C calculation or add only a smaller subset of the fields. But it is to be noted that in this proposal we do have room to consider changes like this without disrupting current installations.

It may also be questionable whether one needs to compute the 16-bit TCP checksum if the new TCP checksum option is present. To avoid a chicken and egg problem, this document proposes that the 16-bit checksum field be zeroed out and included in the CRC 32C checksum as part of the TCP header bit stream. The standardization process may choose a different approach and decide to do both the 16-bit TCP checksum and the CRC 32C checksum, in which case, a method will need to be defined as to the order of checksumming and the fields used in each of the checksum computations.

This document also recommends the use of the CRC-32C when the negotiated Maximum Segment Size (MSS) value is equal or greater than 8948 bytes (excluding frame and TCPIP header bytes), the most common Jumbo Frame size, but does not explicitly recommend the use of CRC-32C for standard Ethernet MTU frames.

The CRC-32C MAY be used even for regular Ethernet MTU frames also if the application so desires for stricter data integrity checking, since CRC-32C can detect more independent bit errors than Ethernet CRC for Ethernet MTU sized packets. The use of CRC-32C can be made settable by the application, by providing a socket option to the application. The provision for an application to enable/disable the use of the new checksum option is left as an API detail of the particular TCP/IP socket layer implementation.

The following section describes two possible approaches to negotiating the proposed 32-bit TCP checksum. The common thread in the two approaches is the use of TCP options to negotiate the use of

this checksum during the connection setup phase. Once the connection is setup, all subsequent packets sent during the connection transfer phase MUST carry the stronger checksum except as described below.

It is also possible that Path MTU discovery causes a connection to reduce the negotiated MSS value post connection establishment. So, during connection establishment, an MSS equal or greater than 9K might have been negotiated along with stronger TCP checksumming, and then later the MSS reduced to be equal to the discovered path MTU. If the reduced MSS value is equal or less than an Ethernet MSS (typically 1460 without other TCP options), then the TCP end point that reduced its MTU may choose to NOT send the TCP checksum option in subsequent data packets. The peer must then rely on the 16-bit TCP checksum for end to end data integrity which is okay since the Ethernet CRC has comparable data integrity checking capability for Ethernet sized packets.

Now, let us discuss the method for computing the CRC 32c value:

The CRC computation uses polynomial division. The TCP header and payload is mapped to a polynomial and the CRC is calculated by dividing the bit stream with the CRC 32C polynomial. Stone et. al. in Appendix B of RFC 4960 [RFC4960] describe a convention for mapping the bytes of the bit stream into the polynomial. The same MUST be adopted for TCP transport too.

3. Negotiating the use of CRC 32C

There are two possible approaches to negotiating the proposed CRC 32C checksum during the TCP connection setup phase.

- A new TCP option
- Using the TCP Alternate Checksum Data Option

The first approach introduces a new TCP option to be negotiated by TCP endpoints during the connection setup phase. It will be of the same format as other defined TCP options and will have Type, Length and Value fields. A new type will be requested from IANA. The length field will be the sum total length of the new TCP checksum option which is 6 bytes. The value field will hold the 32-bit CRC 32C checksum.

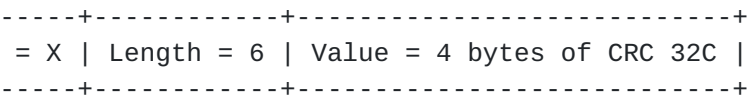
If either one of the peers does not add this option to its TCP options list in its SYN segment, the CRC-32C checksum must not be used by the other peer. Most TCP implementations are written to process the TCP options they recognize and ignore unknown options on

SYN segments so an endpoint that supports the new TCP option can interoperate with an endpoint that does not support the proposed TCP option.

Since we have seen that the 16-bit TCP checksum is insufficient for detecting multiple independent errors for Jumbo frames, this proposal says that a peer supporting this option MUST send the new TCP checksum option if its link MTU is equal or greather than 9K. However, if the remote peer does not recognize the new option, the initiating peer MUST NOT use this TCP extension for the connection transfer phase. If the remote peer recognizes the option and also has a Maximum Segment Size equal to the peer's advertised MSS or a minimum MSS of 9K, it MUST respond with the TCP checksum option. Every subsequent packet from both peers must include this option in the TCP header. The extra overhead for adding this option is minimal for Jumbo frame sized segments and the higher data integrity pays for itself.

Note that all TCP control packets sent after succesfully negotiating this TCP option may carry this TCP option also, although this draft does not mandate it.

TCP CRC Checksum Option.



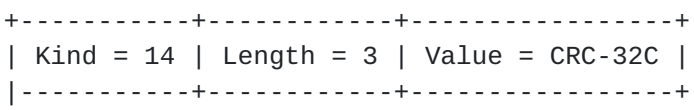
.

Figure 1

The second approach utilizes a pair of existing TCP options called the "TCP Alternate Checksum Options" specified in RFC 1146 [RFC1146]. The current checksum types specified by that option are TCP checksum, 8-bit Fletcher's algorithm and 16-bit Fletcher's algorithm. A new checksum type can be added to this list for CRC-32C checksums. The negotiation rules for selecting the checksum type would follow the rules described in RFC1146. That is, if both SYN segments carry the Alternate Checksum Request option, and both specify the same algorithm, that algorithm must be used for the remainder of the connection. Otherwise, the standard TCP checksum must be used for the entire connection.

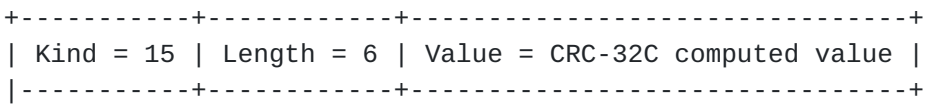
Once the CRC 32C checksum algorithm is negotiated, the TCP Alternate Checksum Data Option is sent whose data will equal 4 bytes for the CRC-32C checksum.

TCP Alternate Checksum Request Option



Here the value for CRC32C would need to be defined, and may possibly be the next undefined value '3', following the definitions for 8-bit and 16-bit fletcher's algorithms.

TCP Alternate Checksum Data Option



The TCP Alternate Checksum Data Option must be sent only during the connection transfer and tear down phase. Again, the 16-bit TCP checksum field must be zeroed out before computing the 32-bit CRC 32C code.

One or more padding bytes may be used when sending any of the above options to align to a 4 or 8 byte boundary for faster parsing on both 32-bit and 64-bit machines.

At this stage of draft development, the author is evaluating and seeking inputs for both approaches.

4. IPv6 Considerations

The TCP extension for CRC 32C can be applied equally to IPv4 and IPv6. The pseudo header for IPv6 includes 128 bit source and destination addresses. This pseudo header, the TCP header and payload MUST be included in the CRC 32C checksum of a TCP/IPv6 segment as there is no IPv6 header checksum.

5. Conclusions and Acknowledgements

This document proposes the use of stronger error detection codes for TCP connections sending Jumbo Frames. It does not provide a solution for UDP based applications. I would also like to thank Tom Kessler (kessler@netapp.com) for his review comments. He specifically pointed out his concerns about the safety of TCP checksum + Ethernet CRC at 40G and 100G speeds with even 9K jumbo frames. He also provided information on the Intel instruction set that can be used to speed up CRC-32c computation. Special thanks to Janet Takami (jtakami@netapp.com) for her comments as well as for pointing out that there is no IPv6 header checksum and so the pseudo header must

be included in the CRC 32c checksum.

6. IANA Considerations

This memo includes a request to IANA for a new Type Number for the new TCP Checksum Option if we do not go with the TCP Alternate Checksum Option. If we go with the TCP Alternate Checksum option, then a new checksum type will need to be defined for CRC 32C, probably after the defined values for Fletcher's 8-bit and 16-bit algorithm types.

7. Security Considerations

The CRC 32C codes can detect unintentional changes to data such as those caused by noise. If an attacker changes the data, it can also change the error-detection code to match the changed data. Hence, these codes are not intended for security purposes.

8. References

8.1. Normative References

[RFC1122] IETF, "Requirements for Internet Hosts -- Communication Layers", October 1989.

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

8.2. Informative References

[Koopman] Koopman, P., "32-Bit Cyclic Redundancy Codes for Internet Applications", 2002.

[Stone] Stone, J., Partridge, C., "When the CRC and TCP Checksum Disagree"

[RFC1146] Zweig, J., Partridge, C., "TCP Alternate Checksum Options" March 1990.

[RFC3385] Sheinwald, D., et. al. "Internet Protocol Small Computer System Interface (iSCSI) Cyclic Redundance Check (CRC)/ Checksum Considerations", September 2002.

[RFC4960] Stewart, R., "Stream Control Transmission Protocol", September 2007.

July 2003.

Author's Address

Anumita Biswas
NetApp, Inc.
495, E. Java Dr
Sunnyvale, CA 95054
USA

Phone: +14088223204
Email: anumita.biswas@netapp.com