Network Working Group
Request for Comments: 2960
Category: Internet Draft

R. Stewart
C. Sharp
Cisco Systems
J. Stone
Stanford
D. Otis
SANlight

January 05, 2002

SCTP Checksum Change
draft-ietf-tsvwg-sctpcsum-01.txt

Status of this Memo

Abstract

SCTP [RFC2960] currently uses an Adler-32 checksum. For small packets, this provides weak protection against the detection of errors. This document changes that checksum and updates SCTP to use a 32 bit CRC checksum.

Table of Contents

1 Introduction

A fundamental weakness has been detected in SCTP's current Adler-32 checksum algorithm [STONE]. One requirement of an effective checksum is that it evenly and smoothly spreads its input packets over the available check bits.

From an email from Jonathan Stone, who analyzed the Adler-32 as part

of his doctoral thesis:

"Briefly, the problem is that, for very short packets, Adler32 is guaranteed to give poor coverage of the available bits. Don't take my word for it, ask Mark Adler. :-).

Adler-32 uses two 16-bit counters, s1 and s2. s1 is the sum of the input, taken as 8-bit bytes. s2 is a running sum of each value of s1. Both s1 and s2 are computed mod-65521 (the largest prime less than 2^16). Consider a packet of 128 bytes. The *most* that each byte can be is 255. There are only 128 bytes of input, so the greatest value which the s1 accumulator can have is 255 * 128 = 32640. So for 128-byte packets, s1 never_ wraps. That is critical. Why?

The key is to consider the distribution of the s1 values, over some distribution of the values of the individual input bytes in each packet. Because s1 never wraps, s1 is simply the sum of the individual input bytes. (even Doug's trick of adding 0x5555 doesn't help here, and an even larger value doesn't really help: we can get at most one mod-565521 reduction).

Given the further assumption that the input bytes are drawn independently from some distribution (they probably aren't: for file system data, it's even worse than that!), the Central Limit Theorem tells us that that s1 will tend to have a normal distribution. That's bad: it tells us that the value of s1 will have hot-spots at around 128 times the mean of the input distribution: around 16k, assuming a uniform distribution. That's bad. We want the accumulator to wrap as many times as possible, so that the resulting sum has as close to a uniform distribution as possible. (I call this "fairness").

So, for short packets, the Adler-32 s1 sum is guaranteed to be unfair. Why is that bad? It's bad because the space of valid packets-- input data, plus checksum values -- is also small. If all packets have checksum values very close to 32640, then the likelihood of even a 'small' error leaving a damaged packet with a valid checksum is higher than if all checksum values are equally likely."

Due to this inherent weakness, exacerbated by the fact that SCTP will first be used as a signaling transport protocol where signaling messages are usually less than 128 bytes, a new checksum algorithm is specified by this document, replacing the current Adler-32 algorithm with CRC-32c.

1.1 Conventions

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD,SHOULD NOT, RECOMMENDED, NOT RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in [RFC2119].

2 Checksum Procedures

The procedures described in section 2.1 of this document MUST be followed, replacing the current checksum defined in [RFC2960]. Furthermore any references within [RFC2960] to Adler-32 MUST be treated

as a reference to CRC-32c. Section 2.1 of this document describes the new calculation and verification procedures that MUST be followed.

2.1 Checksum Calculation

When sending an SCTP packet, the endpoint MUST include in the checksum field the CRC-32c value calculated on the packet, as described below.

After the packet is constructed (containing the SCTP common header and one or more control or DATA chunks), the transmitter MUST do the following:

1) Fill in the proper Verification Tag in the SCTP common header and initialize the checksum field to 0's.

2) Calculate the CRC-32c of the whole packet, including the SCTP common header and all the chunks.

3) Put the resultant value into the checksum field in the common header, and leave the rest of the bits unchanged.

When an SCTP packet is received, the receiver MUST first perform the following:

1) Store the received CRC-32c value,

2) Replace the 32 bits of the checksum field in the received SCTP packet with all '0's and calculate a CRC-32c value of the whole received packet. And,

3) Verify that the calculated CRC-32c value is the same as the received CRC-32c value. If not, the receiver MUST treat the packet as an invalid SCTP packet.

The default procedure for handling invalid SCTP packets is to silently discard them.

The 32 bit CRC is calculated as described for CRC-32c and uses the polynomial code 0x11EDC6F41 (Castagnoli93) or 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 with (reflected) placement. With most serial media, the bits within each byte are shifted out least significant bit first whereas CRC is calculated from most significant to least. To accommodate the serial bit order, a reflected table is used. Reflected means bit 31 becomes bit 0, bit 30 becomes bit 1, etc. This reflected technique also reduces the number of instructions needed for each lookup.

It becomes a minor problem dealing with this unusual reflected value in that both bit and byte order is reversed from that of the CPU. As the bits within each byte are to remain reflected as that is how they are sent out, then ideally only the byte order is adjusted to provide most to least serial presentation. To utilize existing byte placement routines defined for various architectures however, the CRC-32c value will be placed as reflected in network order. This incorrect byte order

placement with respect to the serial sequence eliminates new byte order placement definitions.

To improve leading zero detection, the CRC value is initialized to all one's prior to the packet calculation but is not inverted before being placed. Placement in the SCTP common header and jumbo frames cause variances from the Ethernet CRC algorithm. The [Castagnoli93] polynomial offers error detection enhancements for jumbo frames at the expense of gates. The software table implementations for any 32 bit polynomial has the same overhead however.

3 Acknowledgments

The authors would like to thank the following people that have provided comments and input on the checksum issue:

Ran Atkinson, Stephen Bailey, David Black, Scott Bradner, Mikael Degermark, Laurent Glaude, Klaus Gradischnig, Alf Heidermark, Jacob Heitz, Gareth Kiely, David Lehmann, Allision Mankin, Lyndon Ong, Craig

Partridge, Vern Paxson, Kacheong Poon, Michael Ramalho, David Reed, Ian Rytina, Hanns Juergen Schwarzbauer, Bill Sommerfeld, Michael Tuxen, Jim Williams, Jim Wendt, Michael Welzl, Jonathan Wood, Lloyd Wood, Qiaobing Xie, La Monte Yarroll, Dafna Sheinwald, and Julian Satran, Pat Thaler, Vince Cavanna, Matt Wakeley.

4 Authors' Addresses

Randall R. Stewart
24 Burning Bush Trail.
Crystal Lake, IL 60012
USA

EMail: rrs@cisco.com

Chip Sharp
Cisco Systems Inc.
7025 Kit Creek Road
Research Triangle Park, NC 27709
USA

EMail: chsharp@cisco.com

Jonathan Stone
Room 446, Mail code 9040
Gates building 4A
Stanford, Ca 94305

EMail: jonathan@dsg.stanford.edu

Douglas Otis
800 E. Middlefield

Mountain View, CA 94043
USA

Email dotis@sanlight.net

5 References

[Castagnoli93] Guy Castagnoli, Stefan Braeuer and Martin Herrman

"Optimization of Cyclic Redundancy-Check Codes with 24 and 32 Parity Bits", IEEE Transactions on Communications, Vol. 41, No. 6, June 1993

5.1 Informative References

[STONE] Jonathan Stone "Checksums in the Internet", Doctoral dissertation - August 2001

6 Appendix

Example code using 256 word lookup table.

```
/* Example of the crc table file */
#ifndef __crc32cr_table_h__
#define __crc32cr_table_h__

#define CRC32C_POLY 0x1EDC6F41
#define CRC32C(c,d) (c=(c>>8)^crc_c[(c^(d))&0xFF])
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* Copyright 2001, D. Otis.  Use this program, code or tables    */
/* extracted from it, as desired without restriction.            */
/*                                                               */
/* 32 Bit Reflected CRC table generation for SCTP.               */
/* To accommodate serial byte data being shifted out least       */
/* significant bit first, the table's 32 bit words are reflected */
/* which flips both byte and bit MS and LS positions.  The CRC   */
/* is calculated MS bits first from the perspective of the serial*/
/* stream.  The x^32 term is implied and the x^0 term may also   */
/* be shown as +1.  The polynomial code used is 0x1EDC6F41.      */
/* Castagnoli93                                                  */
/* 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                                     */
/* Guy Castagnoli Stefan Braeuer and Martin Herrman              */
/* "Optimization of Cyclic Redundancy-Check Codes                */
/* with 24 and 32 Parity Bits",                                  */
/* IEEE Transactions on Communications, Vol.41, No.6, June 1993  */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

unsigned long  crc_c[256] =
{
0x00000000L, 0xF26B8303L, 0xE13B70F7L, 0x1350F3F4L,
0xC79A971FL, 0x35F1141CL, 0x26A1E7E8L, 0xD4CA64EBL,
0x8AD958CFL, 0x78B2DBCCL, 0x6BE22838L, 0x9989AB3BL,
0x4D43CFD0L, 0xBF284CD3L, 0xAC78BF27L, 0x5E133C24L,
```

```
0x105EC76FL, 0xE235446CL, 0xF165B798L, 0x030E349BL,
0xD7C45070L, 0x25AFD373L, 0x36FF2087L, 0xC494A384L,
0x9A879FA0L, 0x68EC1CA3L, 0x7BBCEF57L, 0x89D76C54L,
0x5D1D08BFL, 0xAF768BBCL, 0xBC267848L, 0x4E4DFB4BL,
0x20BD8EDEL, 0xD2D60DDDL, 0xC186FE29L, 0x33ED7D2AL,
0xE72719C1L, 0x154C9AC2L, 0x061C6936L, 0xF477EA35L,
0xAA64D611L, 0x580F5512L, 0x4B5FA6E6L, 0xB93425E5L,
0x6DFE410EL, 0x9F95C20DL, 0x8CC531F9L, 0x7EAEB2FAL,
0x30E349B1L, 0xC288CAB2L, 0xD1D83946L, 0x23B3BA45L,
0xF779DEAEL, 0x05125DADL, 0x1642AE59L, 0xE4292D5AL,
0xBA3A117EL, 0x4851927DL, 0x5B016189L, 0xA96AE28AL,
0x7DA08661L, 0x8FCB0562L, 0x9C9BF696L, 0x6EF07595L,
0x417B1DBCL, 0xB3109EBFL, 0xA0406D4BL, 0x522BEE48L,
0x86E18AA3L, 0x748A09A0L, 0x67DAFA54L, 0x95B17957L,
0xCBA24573L, 0x39C9C670L, 0x2A993584L, 0xD8F2B687L,
0x0C38D26CL, 0xFE53516FL, 0xED03A29BL, 0x1F682198L,
0x5125DAD3L, 0xA34E59D0L, 0xB01EAA24L, 0x42752927L,
0x96BF4DCCL, 0x64D4CECFL, 0x77843D3BL, 0x85EFBE38L,
0xDBFC821CL, 0x2997011FL, 0x3AC7F2EBL, 0xC8AC71E8L,
0x1C661503L, 0xEE0D9600L, 0xFD5D65F4L, 0x0F36E6F7L,
0x61C69362L, 0x93AD1061L, 0x80FDE395L, 0x72966096L,
0xA65C047DL, 0x5437877EL, 0x4767748AL, 0xB50CF789L,
0xEB1FCBADL, 0x197448AEL, 0x0A24BB5AL, 0xF84F3859L,
0x2C855CB2L, 0xDEEEDFB1L, 0xCDBE2C45L, 0x3FD5AF46L,
0x7198540DL, 0x83F3D70EL, 0x90A324FAL, 0x62C8A7F9L,
0xB602C312L, 0x44694011L, 0x5739B3E5L, 0xA55230E6L,
0xFB410CC2L, 0x092A8FC1L, 0x1A7A7C35L, 0xE811FF36L,
0x3CDB9BDDL, 0xCEB018DEL, 0xDDE0EB2AL, 0x2F8B6829L,
0x82F63B78L, 0x709DB87BL, 0x63CD4B8FL, 0x91A6C88CL,
0x456CAC67L, 0xB7072F64L, 0xA457DC90L, 0x563C5F93L,
0x082F63B7L, 0xFA44E0B4L, 0xE9141340L, 0x1B7F9043L,
0xCFB5F4A8L, 0x3DDE77ABL, 0x2E8E845FL, 0xDCE5075CL,
0x92A8FC17L, 0x60C37F14L, 0x73938CE0L, 0x81F80FE3L,
0x55326B08L, 0xA759E80BL, 0xB4091BFFL, 0x466298FCL,
0x1871A4D8L, 0xEA1A27DBL, 0xF94AD42FL, 0x0B21572CL,
0xDFEB33C7L, 0x2D80B0C4L, 0x3ED04330L, 0xCCBBC033L,
0xA24BB5A6L, 0x502036A5L, 0x4370C551L, 0xB11B4652L,
0x65D122B9L, 0x97BAA1BAL, 0x84EA524EL, 0x7681D14DL,
0x2892ED69L, 0xDAF96E6AL, 0xC9A99D9EL, 0x3BC21E9DL,
0xEF087A76L, 0x1D63F975L, 0x0E330A81L, 0xFC588982L,
0xB21572C9L, 0x407EF1CAL, 0x532E023EL, 0xA145813DL,
0x758FE5D6L, 0x87E466D5L, 0x94B49521L, 0x66DF1622L,
0x38CC2A06L, 0xCAA7A905L, 0xD9F75AF1L, 0x2B9CD9F2L,
0xFF56BD19L, 0x0D3D3E1AL, 0x1E6DCDEEL, 0xEC064EEDL,
0xC38D26C4L, 0x31E6A5C7L, 0x22B65633L, 0xD0DDD530L,
0x0417B1DBL, 0xF67C32D8L, 0xE52CC12CL, 0x1747422FL,
0x49547E0BL, 0xBB3FFD08L, 0xA86F0EFCL, 0x5A048DFFL,
0x8ECEE914L, 0x7CA56A17L, 0x6FF599E3L, 0x9D9E1AE0L,
0xD3D3E1ABL, 0x21B862A8L, 0x32E8915CL, 0xC083125FL,
0x144976B4L, 0xE622F5B7L, 0xF5720643L, 0x07198540L,
```

```
    0x590AB964L, 0xAB613A67L, 0xB831C993L, 0x4A5A4A90L,
    0x9E902E7BL, 0x6CFBAD78L, 0x7FAB5E8CL, 0x8DC0DD8FL,
    0xE330A81AL, 0x115B2B19L, 0x020BD8EDL, 0xF0605BEEL,
    0x24AA3F05L, 0xD6C1BC06L, 0xC5914FF2L, 0x37FACCF1L,
```



```
    0x69E9F0D5L, 0x9B8273D6L, 0x88D28022L, 0x7AB90321L,
    0xAE7367CAL, 0x5C18E4C9L, 0x4F48173DL, 0xBD23943EL,
    0xF36E6F75L, 0x0105EC76L, 0x12551F82L, 0xE03E9C81L,
    0x34F4F86AL, 0xC69F7B69L, 0xD5CF889DL, 0x27A40B9EL,
    0x79B737BAL, 0x8BDCB4B9L, 0x988C474DL, 0x6AE7C44EL,
    0xBE2DA0A5L, 0x4C4623A6L, 0x5F16D052L, 0xAD7D5351L,
    };

    #endif

    /* Example of table build routine */

    #include <stdio.h>
    #include <stdlib.h>

    #define OUTPUT_FILE   "crc32cr.h"
    #define CRC32C_POLY    0x1EDC6F41L
    #define CRC_TYPE  "\
    /* Castagnoli93                                                  */\n\
    /* x^32+x^28+x^27+x^26+x^25+x^23+x^22+x^20+x^19+x^18+x^14+x^13+  */\n\
    /* x^11+x^10+x^9+x^8+x^6+x^0                                     */\n\
    /* Guy Castagnoli Stefan Braeuer and Martin Herrman              */\n\
    /* \"Optimization of Cyclic Redundancy-Check Codes                */\n\
    /* with 24 and 32 Parity Bits\",                                  */\n\
    /* IEEE Transactions on Communications, Vol.41, No.6, June 1993  */\n"

    FILE *tf;

    unsigned long
      reflect_32 (unsigned long b)
    {
      int i;
      unsigned long rw = 0L;

      for (i = 0; i < 32; i++)
        {
          if (b & 1)
            rw |= 1 << (31 - i);
          b >>= 1;
        }
```

```
    return (rw);
  }

  unsigned long
    build_crc_table (int index)
  {
    int i;
    unsigned long rb;

    rb = reflect_32 (index);

    for (i = 0; i < 8; i++)
      {
        if (rb & 0x80000000L)
```



```
         rb = (rb << 1) ^ CRC32C_POLY;
        else
         rb <<= 1;
      }
    return (reflect_32 (rb));
  }

  main ()
  {
    int i;

    printf ("\nGenerating CRC-32c table file <%s>\n", OUTPUT_FILE);
    if ((tf = fopen (OUTPUT_FILE, "w")) == NULL)
      {
        printf ("Unable to open %s\n", OUTPUT_FILE);
        exit (1);
      }
    fprintf (tf, "#ifndef __crc32cr_table_h__\n");
    fprintf (tf, "#define __crc32cr_table_h__\n\n");
    fprintf (tf, "#define CRC32C_POLY 0x%08lX\n", CRC32C_POLY);
    fprintf (tf, "#define CRC32C(c,d) (c=(c>>8)^crc_c[(c^(d))&0xFF])\n");
    fprintf (tf, "\
  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */\n\
  /* Copyright 2001, D. Otis.  Use this program, code or tables    */\n\
  /* extracted from it, as desired without restriction.            */\n\
  /*                                                               */\n\
  /* 32 Bit Reflected CRC table generation for SCTP.               */\n\
  /* To accommodate serial byte data being shifted out least       */\n\
  /* significant bit first, the table's 32 bit words are reflected */\n\
  /* which flips both byte and bit MS and LS positions.  The CRC   */\n\
```

```
/* is calculated MS bits first from the perspective of the serial*/\n\
/* stream.  The x^32 term is implied and the x^0 term may also   */\n\
/* be shown as +1.  The polynomial code used is 0x%08lX.      */\n%s\
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */\n"
               , CRC32C_POLY, CRC_TYPE);

  fprintf (tf, "\nunsigned long  crc_c[256] =\n{\n");
  for (i = 0; i < 256; i++)
    {
      fprintf (tf, "0x%08lXL, ", build_crc_table (i));
      if ((i & 3) == 3)
        fprintf (tf, "\n");
    }

  fprintf (tf, "};\n\n#endif\n");

  if (fclose (tf) != 0)
    printf ("Unable to close <%s>." OUTPUT_FILE);
  else
    printf ("\nThe CRC-32c table has been written to <%s>.\n",
      OUTPUT_FILE);
}
```



```
/* Example of crc insertion */

#include "crc32cr.h"


int
insert_crc32(unsigned char *buffer, unsigned int length)
{
SCTP_message *message;
unsigned int i;
unsigned long crc32 = ~0L;

/* check packet length */
if (length > NMAX  || length < NMIN)
  return -1;

message = (SCTP_message *) buffer;
message->common_header.checksum = 0L;
```

```
for (i = 0; i < length; i++)
{
  CRC32C(crc32, buffer[i]);
}

/* and insert it into the message */
message->common_header.checksum = htonl(crc32);
    return 1;
}

/* Example of crc validation */
/* Test of 32 zeros should yield 0x756EC955 placed in network order */
/* 13 zeros followed by byte values of 1 - 0x1f should yield
/* 0x5b988D47 */

int
validate_crc32(unsigned char *buffer, unsigned int length)
{
SCTP_message *message;
unsigned int i;
unsigned long original_crc32;
unsigned long crc32 = ~0L;

/* check packet length */
if (length > NMAX  || length < NMIN)
  return -1;

/* save and zero checksum */
message = (SCTP_message *) buffer;
original_crc32 = ntohl(message->common_header.checksum);
message->common_header.checksum = 0L;

for (i = 0; i < length; i++)
{
```



```
  CRC32C(crc32, buffer[i]);
}

  return ((original_crc32 == crc32)? 1 : -1);
}
```

Full Copyright Statement

Funding for the RFC Editor function is currently provided by the Internet Society.