Network Working Group
Request for Comments: 3309
Updates: 2960
Category: Standards

J. Stone
Stanford
R. Stewart
Cisco Systems
D. Otis
SANlight
September 2002

Изменение контрольных сумм SCTP

Stream Control Transmission Protocol (SCTP) Checksum Change

Статус документа

Этот документ является спецификацией стандарта Internet, предназначенного для сообщества Internet, и служит приглашением к дискуссии в целях развития протокола. Сведения о текущем состоянии стандартизации протокола можно найти в документе «Internet Official Protocol Standards» (STD 1). Документ может распространяться без ограничений.

Авторские права

Тезисы

Протокол SCTP[1] в настоящее время использует контрольные суммы Adler-32. Для небольших пакетов алгоритм Adler-32 не обеспечивает надежного детектирования ошибок. Этот документ меняет алгоритм расчета контрольных сумм и переводит SCTP на использование 32-битовых контрольных сумм CRC.

Оглавление

1 Введение

В используемом протоколом SCTP алгоритме контрольных сумм Adler-32 были обнаружены существенные недостатки [STONE]. Этот документ заменяет алгоритм Adler-32, определенный в [RFC 2960] для контрольных сумм. Следует отметить, что механизма плавного перехода к новым контрольным суммам не предлагается. Возлагается надежда на быстрый переход приложений к новым контрольным суммам а использование старого алгоритма отменяется.

Одним из требований к эффективному алгоритму контрольных сумм является равномерное распределение входной информации по битам контрольной суммы.

Ниже приведена цитата из диссертации Jonathan Stone, посвященная анализу алгоритма Adler-32.

> Говоря кратко, проблема заключается в том, что для очень коротких пакетов Adler32 не гарантирует хорошего покрытия всех доступных битов. Не верите мне на слово — спросите Mark Adler. :-)
>
> Adler-32 использует два 16-битовых счетчика s1 и s2. В s1 содержится сумма входных данных, рассматриваемых, как 8-битовые байты, а s2 содержит сумму значений s1. Значения счетчиков s1 и s2 рассчитываются по модулю 65521 (наибольшее простое число, которое меньше 2^{16}). Рассмотрим пакет размером 128 байтов. Наибольшим значением каждого байта является 255. На входе имеется лишь 128 байтов, поэтому максимальное значение счетчика s1 составит 255 * 128 = 32640. Следовательно, для 128-байтовых пакетов s1 никогда не достигает максимального значения. Это критично. Почему?
>
> Важно рассмотреть распределение значений s1 для некоторого распределения значений отдельных байтов в каждом пакете. Поскольку s1 никогда не достигает максимума, значение s1 является обычной суммой отдельных входных байтов (уловка Doug с добавлением 0x5555 или даже добавление большего значения не помогает — мы не получим более одного перехода через максимум).
>
> Далее, в предположении что входные байты берутся независимо из некого распределения (для реальных данных из файлов все будет еще хуже), в соответствии с теоремой центрального предела s1 будет иметь тенденцию к центральному распределению. Это плохо и говорит нам, что s1 будет иметь примерно в 128 раз больше «горячих точек» нежели усредненное входное распределение — около 16k в предположении однородного распределения. Это плохо. Мы ждем от аккумулятора максимального числа переходов через максимум, чтобы получаемая в

[1]Stream Control Transmission Protocol — протокол управления потоковой передачей.

результате сумма имела максимально близкое к однородному распределение (я назвал это «беспристрастностью»).

По этой причине сумма Adler-32 s1 явно не подходит для коротких пакетов. Что в этом плохого? Это плохо потому, что пространство корректных пакетов (входные данные и значения контрольных сумм) также весьма мало. Если все пакеты будут иметь контрольные суммы около 32640, вероятность того, что «маленькая» ошибка в пакет не приведет к некорректности контрольной суммы будет значительно меньше, чем при однородном распределении.

С учетом отмеченного недостатка, а также того факта, что SCTP будет применяться, прежде всего, в качестве сигнального транспортного протокола, а размер сигнальных сообщений обычно меньше 128 байтов, данный документ указывает замену алгоритма Adler-32 на CRC-32c.

1.1 Соглашения о терминах

Ключевые слова **необходимо** (MUST), **недопустимо** (MUST NOT), **требуется** (REQUIRED), **нужно** (SHALL), **не нужно** (SHALL NOT), **следует** (SHOULD), **не следует** (SHOULD NOT), **рекомендуется** (RECOMMENDED), **возможно** (MAY), **необязательно** (OPTIONAL) в данном документе должны интерпретироваться в соответствии с [RFC 2119].

Порядок битов определен в [RFC1700].

2 Процедуры

Описанные в параграфе 2.1 этого документа процедуры **должны** выполняться взамен используемых в настоящее время контрольных сумм, которые были определены в [RFC2960].

Кроме того, все ссылки [RFC2960] на Adler-32 **должны** быть заменены на CRC-32c. В параграфе 2.1 данного документа описаны новые проедуры расчета и проверки контрольных сумм, которые **должны** применяться.

2.1 Расчет контрольных сумм

При передаче пакета SCTP конечная точка **должна** для обеспечения возможности контроля целостности данных включить в пакет значение контрольной суммы CRC32c в соответствии с приведенным ниже описанием.

После создания пакета (содержащего общий заголовок SCTP и один или несколько управляющих блоков или блоков DATA), отправитель **должен** выполнить перечисленные ниже операции.

1) Заполнить поле Verification Tag общего заголовка SCTP и установить нулевое значение в поле контрольной суммы.
2) Рассчитать контрольную сумму CRC32c с включением общего заголовка SCTP и всех блоков из пакета.
3) Поместить полученное значение в поле контрольной суммы общего заголовка, не изменяя остальных битов.

При получении пакета SCTP принимающая конечная точка **должна** сначала проверить значение контрольной суммы CRC32c в заголовке, как описано ниже.

1) Сохранить полученное в пакете значение контрольной суммы CRC32c.
2) Заменить 32-битовое поле контрольной суммы принятого пакета SCTP значением 0 и рассчитать контрольную сумму CRC32c для измененного пакета.
3) Сравнить сохраненное значение (из пакета) CRC32c с контрольной суммой, полученной в результате расчета. Если значения не совпадут, получатель **должен** трактовать принятый пакет как некорректный.

По умолчанию некорректные пакеты SCTP отбрасываются без уведомления.

Аппаратным реализациям **следует** обеспечивать возможность программной проверки контрольных сумм.

Мы определяем «отраженное значение», как значение с порядком битов, обратным по отношению к используемому в машине. 32-битовое значение CRC рассчитывается, как описано для CRC-32c и использует полиномиальный код 0x11EDC6F41 (Castagnoli93) или $x^{32}+x^{28}+x^{27}+x^{26}+x^{25}+x^{23}+x^{22}+x^{20}+x^{19}+x^{18}+x^{14}+x^{13}+x^{11}+x^{10}+x^{9}+x^{8}+x^{6}+x^{0}$. Значение CRC рассчитывается с помощью процедуры, похожей на ETHERNET CRC [ITU32], которая изменена с учетом применения на транспортном уровне.

Расчет CRC использует полиномиальное деление. Битовая строка сообщения (M) преобразуется в полином M(X) и значение CRC рассчитывается по M(X) с использованием полиномиальной арифметики [Peterson 72].

При использовании CRC на канальном уровне полином строится с использованием естественного порядка битов - первый бит сигнала в линии является коэффициентов старшего порядка. Поскольку SCTP является протоколом транспортного уровня, он не может знать порядка передачи битов в физическую среду. Более того, на разных участках пути между конечными точками SCTP может на канальном уровне применяться разный порядок битов.

Следовательно, требуется соглашение для отображения транспортных сообщений SCTP на полиномы с целью расчета CRC. При отображении сообщений SCTP на полиномы сначала берется старший байт, но в каждом байте биты берутся, начиная с младшего. Первый байт сообщения обеспечивает восемь старших коэффициентов. В каждом байте младший бит SCTP дает самый старший коэффициент полинома в рамках данного байта, а старший бит SCTP — младший коэффициент в рамках байта (такой порядок иногда называют отраженным — mirrored или reflected [Williams93]). Полиномы CRC преобразуются обратно в значения байтов транспортного уровня SCTP с помощью соответствующего отображения.

Значение CRC для транспортного уровня SCTP следует рассчитывать в соответствии с приведенным ниже описанием.

- Входными данными CRC является поток байтов с номерами 0 - N-1.
- Байтовый поток транспортного уровня отображается на полиномиальное значение. PDU размером N байтов с номерами j от 0 до N-1 рассматривается, как коэффициенты полинома M(x) порядка 8N-1 и бит 0 байта j будет коэффициентом $x^{(8(N-j)-8)}$, а бит 7 этого байта - $x^{(8(N-j)-1)}$.

- Оставшаяся часть регистра CRC заполняется единицами и рассчитывается значение CRC с помощью умножения на x^{32} и деления на полином CRC.
- Полином умножается на x^{32} и делится на G(x), что порождает полином степени не более 31, образующий оставшуюся часть R(x).
- Коэффициенты R(x) рассматриваются, как 32-битовая последовательность.
- Последовательность битов дополняется и результат является полиномом CRC.
- Полином CRC отображается обратно на байты транспортного уровня SCTP. Коэффициент x^{31} дает значение биты 7 в байте SCTP 0, а коэффициент x^{24} дает значение бита 0 в байте 0. Коэффициент x^{7} дает значение бита 7 в байте 3, а коэффициент x^{0} — значение бита 0 в байте 3. Результирующая 4-байтовая последовательность является последовательностью транспортного уровня для 32-битовой контрольной суммы SCTP.

Примечание для разработчиков. В стандартах, книгах и литературе от производителей для CRC зачастую приводятся иные формулировки, где используется регистр для хранения остатка алгоритма деления long-division, инициализируемый нулями, а не 1 и первые 32 бита сообщения дополняются. Алгоритм long-division, используемый в нашей формулировке совмещает начальное умножение на 2^{32} и «длинное деление» в одной операции. Для таких алгоритмов и сообщений, размер которых превышает 64 бита, две спецификации полностью эквивалентны. Обеспечение эквивалентности является одной из целей данного документа.

Следует предупредить разработчиков SCTP о том, что в литературе можно найти обе спецификации и иногда без ограничений для алгоритма long-division. Выбор формулировки в этом документе обусловлен возможностью применения не только для SCTP, когда тот же алгоритм CRC может применяться для сообщений меньше 64 битов.

Если SCTP не может следовать применению CRC на канальном уровне, значения CRC можно рассчитать для битового потока канального уровня. Первый бит отображается на коэффициент старшего порядка и т. д., вплоть до последнего бита канального уровня, отображаемого на коэффициент младшего порядка. Значение CRC будет передаваться сразу же после входного сообщения, как «трейлер» канального уровня. Результирующий битовый поток канального уровня будет иметь форму $(M(X)x) * x^{32}) + (M(X)*x^{32}))/G(x)$, which is divisible by G(X). There would thus be a constant CRC remainder for 'good' packets. Однако, с учетом широкого распространения реализаций RFC 2960, обсуждение в IETF показало, что преимущества использования «трейлеров» CRC не компенсируют издержек, связанный со значительным изменением работы протокола. Кроме того, пакеты, воспринимаемые по контрольной сумме в заголовке SCTP, полностью соответствуют пакетам, воспринимаемым измененной процедурой на основе «трейлера» CRC, когда контрольная в общем заголовке SCTP контрольная сумма устанавливается в 0 при передаче и принимается, как 0.

Можно несколько снизить вычислительные издержки, проверяя ассоциацию по значению Verification Tag до расчета контрольной суммы, чтобы не рассчитывать контрольные суммы для пакетов с некорректными тегами. Исключением из этого правила являются блоки INIT и некоторые обмены SHUTDOWN-COMPLETE, а также устаревшие блоки COOKIE ECHO. Однако в этих случаях пакеты достаточно малы и расчет контрольных сумм не требует значительных ресурсов.

3 Вопросы безопасности

Вопросы безопасности, рассмотренные в RFC 2960, применимы и к протоколу с новыми контрольными суммами.

4 Согласование с IANA

Этот документ не требует какого-либо согласования с IANA.

5 Благодарности

Авторы благодарят за комментарии и вклад в разработку документа Mark Adler, 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, Chip Sharp, Bill Sommerfeld, Michael Tuexen, Jim Williams, Jim Wendt, Michael Welzl, Jonathan Wood, Lloyd Wood, Qiaobing Xie, La Monte Yarroll.

Особая благодарность Dafna Scheinwald, Julian Satran, Pat Thaler, Matt Wakeley и Vince Cavanna,за выбор критериев для полинома и проверку полиномов CRC, в частности CRC-32c [Castagnoli93].

Отдельная благодарность Mr. Ross Williams за его документ [Williams93]. Это неформальное рассмотрение программных аспектов CRC существенно помогло авторам, недостаточно знакомым с расчетами CRC. Более формальные трактовки [Blahut 94] и [Peterson 72] также оказались полезными.

6 Литература

[Castagnoli93] G. Castagnoli, S. Braeuer and M. Herrman, "Optimization of Cyclic Redundancy-Check Codes with 24 and 32 Parity Bits", IEEE Transactions on Communications, Vol. 41, No. 6, June 1993

[McKee75] H. McKee, "Improved {CRC} techniques detects erroneous leading and trailing 0's in transmitted data blocks", Computer Design Volume 14 Number 10 Pages 102-4,106, October 1975

[RFC1700] Reynolds, J. and J. Postel, "ASSIGNED NUMBERS", RFC 1700[1], October 1994.

[RFC2026] Bradner, S., "The Internet Standards Process – Revision 3", BCP 9, RFC 2026, October 1996.

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

[RFC2960] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L. and V. Paxson, "Stream Control Transmission Protocol," RFC 2960, October 2000.

[1]В соответствии с RFC 3232 этот документ утратил силу и заменен базой данных на сайте IANA. *Прим. перев.*

[ITU32] ITU-T Recommendation V.42, "Error-correcting procedures for DCEs using asynchronous-to-synchronous conversion", section 8.1.1.6.2, October 1996.

6.1 Дополнительная литература[1]

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

[Williams93] Williams, R., "A PAINLESS GUIDE TO CRC ERROR DETECTION ALGORITHMS" - Internet publication, August 1993, http://www.geocities.com/SiliconValley/Pines/8659/crc.htm.

[Blahut 1994] R.E. Blahut, Theory and Practice of Error Control Codes, Addison-Wesley, 1994.

[Easics 2001] http://www.easics.be/webtools/crctool. Online tools for synthesis of CRC Verilog and VHDL.

[Feldmeier 95] David C. Feldmeier, Fast software implementation of error detection codes, IEEE Transactions on Networking, vol 3 no 6, pp 640-651, December, 1995.

[Glaise 1997] R. J. Glaise, A two-step computation of cyclic redundancy code CRC-32 for ATM networks, IBM Journal of Research and Development} vol 41 no 6, 1997. http://www.research.ibm.com/journal/rd/416/glaise.html.

[Prange 1957] E. Prange, Cyclic Error-Correcting codes in two symbols, Technical report AFCRC-TN-57-103, Air Force Cambridge Research Center, Cambridge, Mass. 1957.

[Peterson 1972] W. W. Peterson and E.J Weldon, Error Correcting Codes, 2nd. edition, MIT Press, Cambridge, Massachusetts.

[Shie2001] Ming-Der Shieh et. al, A Systematic Approach for Parallel CRC Computations. Journal of Information Science and Engineering, Vol.17 No.3, pp.445-461

[Sprachman2001] Michael Sprachman, Automatic Generation of Parallel CRC Circuits, IEEE Design & Test May-June 2001

Приложение

Это приложение **не** является частью стандарта и представлено лишь для информации.

Ожидаемое развертывание SCTP может использовать различающиеся на несколько порядков скорости каналов — от линий сотовой связи со скоростями в десятки килобит/сек для локальных сетей с десятками гигабит/сек. Разработчикам SCTP следует принимать во внимание скорость канала и выбирать из множества реализаций CRC ту, которая соответствует требованиям в части размера, издержек и пропускной способности. Известно множество методов расчета CRC. В этом приложении рассмотрены лишь некоторые из них для того, чтобы дать общее представления.

Методы CRC основаны на ранних работах Prange (вередина 1950-х) [Prange 57]. Теория CRC и выбор порождающего многочлена основывается на полях Галуа (Galois) [Blahut 94] или идеях алгебры циклических кодов [Peterson 72].

Одним из простейших методов является последовательная аппаратная реализация, в которой порождающий многочлен образован отводами (tap) линейного регистра сдвига с обратной связью (LSFR[2]). Расчеты LSFR в основном описаны Prange. Имеются средства, в которых порождающий многочлен CRC выдает синтезируемый код Verilog для оборудования CRC [Easics 2001].

Поскольку LSFR не обеспечивают достаточного масштабирования по скорости, было разработано множество иных методов. Один из них основан на том, что делитель полиномиального «длинного деления» (G) известен заранее. Это позволяет заранее рассчитать таблицы, дающие остаток полинома для множества входных битов (обычно 2, 4 или 8 битов в один прием). Этот метод можно реализовать как программными, так и аппаратными средствами. Программы для расчета таблиц просмотра, дающие результат в 2, 4 или 8 битов, распространяются свободно [Williams93].

Для мультигигабитных каналов описанных выше методов может оказаться недостаточно. Одним из методов расчета CRC при скоростях OC-48 является «двухэтапный расчет» (two-stage) [Glaise 1997]. Здесь один из сомножителей (G(x)) G(x)H(x) выбирается так, чтобы минимизировать число ненулевых коэффициентов или вес произведения G(x)H(x). Малый вес произведения полиномов делает его подходящим для эффективных аппаратных реализаций на базе деления на константу. Этот первый этап дает в качестве результата M(x)/(G(x)H(x)). На втором этапе этот результат делится на H(x), давая (M(x)/(G(x)H(x)))/H(x). Если H(x) и G(x) являются взаимно простыми, это дает M(x)/G(x). дальнейшее развитие этого метода можно найти в [Shie2001] и [Sprachman2001].

В литературе упоминается множество программных реализаций CRC. Одним из вариантов является использование оптимизированного ассемблерного кода для прямого полиномиального деления. В работе [Feldmeier 95] показано, что для полиномов с малым весом оптимизированная полиномиальная арифметика обеспечивает более высокую производительность, нежели поиск в таблице. Даже в алгоритме поиска по таблице размер таблицы можно оптимизировать путем использования кэшированных отпечатков (для систем с ограниченным пространством) или минимизации общего размера.

Разработчикам следует принимать во внимание порядок битов, описанный в разделе 2. Порядок битов в байтах для расчета CRC в SCTP таков, что младший бит каждого байта является старшим коэффициентом полинома и наоборот. «Отраженный» порядок битов SCTP CRC соответствует порядку передачи битов в среде Ethernet и других последовательных линиях, но обратен по отношению к традиционному порядку битов в Internet.

Далее описан один из методов обращения битов. Представим себе аппаратную реализацию с порядком битов CRC и выполним обращение битов слева направо (отражение) для всего алгоритма (вопрос порядка байтов в словах пока отложим). После этого рассчитаем зеркальное отражение битов программным путем. CRC от алгоритма «зеркального отражения» будет иметь обратный по сравнению с аппаратной реализацией. Когда байты передаются в среду канального уровня используется порядок битов, обратный по отношению к используемому в CPU хоста. Преобразование каждого байта «отраженного» CRC в последовательный формат будут заново обращать порядок битов и с результате биты будут упорядочены в нужном для среды виде.

[1] В оригинале этот параграф имеет ошибочный номер 7.1. *Прим. перев.*

[2] Linear feedback shift register.

Ниже приведен (ненормативный) пример кода, заимствованный из генератора CRC с открытым кодом [WILLIAMS93], использующего метод «отражения» и таблицу для SCTP CRC32c с 256 записями по 32 бита в каждой. Этот метод не слишком быстрый и не слишком медленный по сравнению с поиском в таблицах CRC, но его преимуществом является возможность работы с процессами, использующими порядок big-endian и little-endian, при просмотре общих таблиц (с хост-порядком), а также использование лишь предопределенных операций ntohl() и htonl(). Код несколько отличается от [WILLIAMS93] для обеспечения переносимости между архитектурами big-endian и little-endian (отметим, что в тех случаях, когда известно, что целевая архитектура использует порядок little-endian, финальные операции bit-reversal и byte-reversal могут быть объединены).

```
/**********************************************************/
/* Для генератора таблиц Ross Williams устанавливаются    */
/* значения TB_WIDTH=4, TB_POLLY=0x1EDC6F41, TB_REVER=TRUE */
/* Для прямого расчета Mr. Williams используются значения */
/* cm_width=32, cm_poly=0x1EDC6F41, cm_init=0xFFFFFFFF,   */
/* cm_refin=TRUE, cm_refot=TRUE, cm_xorort=0x00000000     */
/**********************************************************/

/* Пример файла с таблицей crc */
#ifndef __crc32cr_table_h__
#define __crc32cr_table_h__

#define CRC32C_POLY 0x1EDC6F41
#define CRC32C(c,d) (c=(c>>8)^crc_c[(c^(d))&0xFF])

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

 /* Пример программы построения таблицы */

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

#define OUTPUT_FILE   "crc32cr.h"
#define CRC32C_POLY    0x1EDC6F41L
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, "\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);
}

/* Пример вставки crc */

#include "crc32cr.h"

unsigned long
generate_crc32c(unsigned char *buffer, unsigned int length)
{
  unsigned int i;
  unsigned long crc32 = ~0L;
  unsigned long result;
  unsigned char byte0,byte1,byte2,byte3;

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

  /*  в result храниться «негативный» остаток полинома,
   *  поскольку таблица и алгоритм являются «зеркальными [williams95].
   *  Т. е., result имеет такое же значение, как будто мы отобразили сообщение
   *  на полином, рассчитали остаток полинома для хостового порядка битов
   *  выполнили финальную смену знака (negation) и сквозное обращение битов.
   *  Отметим, что 32-битовое обращение битов идентично 4 восьмибитовым обращениям
   *  с последующим обращением порядка байтов. На машинах little-endian
   *  такое обращение байтов и финальная операция ntohl cancel не нужны.
   */

  byte0 = result & 0xff;
  byte1 = (result>>8) & 0xff;
  byte2 = (result>>16) & 0xff;
  byte3 = (result>>24) & 0xff;

  crc32 = ((byte0 << 24) |
           (byte1 << 16) |
           (byte2 << 8)  |
          byte3);
  return ( crc32 );
}

int
insert_crc32(unsigned char *buffer, unsigned int length)
{
  SCTP_message *message;
  unsigned long crc32;
  message = (SCTP_message *) buffer;
  message->common_header.checksum = 0L;
  crc32 = generate_crc32c(buffer,length);
  /* and insert it into the message */
  message->common_header.checksum = htonl(crc32);
  return 1;
}

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

  /* сохраним и обнулим контрольную сумму */
  message = (SCTP_message *) buffer;
  original_crc32 = ntohl(message->common_header.checksum);
  message->common_header.checksum = 0L;
  crc32 = generate_crc32c(buffer,length);
  return ((original_crc32 == crc32)? 1 : -1);
}
```

Адреса авторов

Jonathan Stone

Room 446, Mail code 9040

Gates building 4A

Stanford, Ca 94305

EMail: jonathan@dsg.stanford.edu

Randall R. Stewart

24 Burning Bush Trail.

Crystal Lake, IL 60012

USA

EMail: rrs@cisco.com

Douglas Otis

800 E. Middlefield

Mountain View, CA 94043

USA

EMail: dotis@sanlight.net

Перевод на русский язык

Николай Малых

nmalykh@gmail.com

Полное заявление авторских прав

Подтверждение

Финансирование функций RFC Editor обеспечено Internet Society.