CRC64 Algorithm Analysis and Verification[1]

Petr Hlávka, Tomáš Kratochvíla, Vojtěch Řehák, David Šafránek, Pavel Šimeček, and Tomáš Vojnar

December 15, 2005

1 Abstract

This work analyzes the use of a CRC64 algorithm as a hashing function in the Netflow project. We describe the basis of Cyclic Redundancy Check (CRC) algorithms and consider properties like collision probability, Hamming distance, and quality of distribution, which are crucial for hashing functions. Lower or upper bounds of these properties are described mathematically. However, to give more precise numbers to hardware designers, we also try to find them using model checking method.

2 Introduction

The Cyclic Redundancy Check (CRC) algorithms were originally developed for detection of line transmission errors. They were designed to be fast and easy to implement in hardware. Thanks to their low hardware requirements, and modulo function behavior, they are also widely used as simple hashing functions in non-cryptographic applications.

In the Liberouter project [LibWWW], we use various types of CRC functions to generate checksums. The same well-tested hardware design was also used for hashing in the Netflow probe. The Netflow probe is a passive network IP flow monitoring device. Information about each monitored IP flow (connection) has to be stored in the internal memory. CRC is used for unique flow identification and also for determining the address in the memory, where the additional flow information is stored. This paper tries to find out the quality of the flow identification uniqueness (hash collision probability). The distribution quality of memory addresses was measured by statistical testing [StatTest] and will be part of a simulation model of the Netflow probe.

---

3   Cyclic Redundancy Codes (CRC)

This section briefly describes mathematical background of CRC algorithms and also introduces widely used fast software and hardware implementation.

3.1   Principle of CRC

CRCs are based on division in a ring of polynomials over integers modulo 2 [MathWorldCRC], [CompNets]. In this modular arithmetic, coefficients of polynomials are represented by only one bit and any other values than 0 or 1 are wrapped around by the modulo 2 operation. Any string of bits can be interpreted as a sequence of polynomial coefficients. The CRC is defined as a sequence of coefficients appearing in the remainder polynomial. The remainder polynomial is gotten from the input string of bits dividing it by the so called generating polynomial.

Addition:

$$(x^3 + x^2 + x + 1) + (x^2 + 1) = x^3 + 2x^2 + x + 1 = x^3 + x + 1$$

Substraction:

$$(x^3 + x + 1) - (x^2 + 1) = x^3 - x^2 + x = x^3 + x^2 + x$$

Multiplication:

$$(x^2 + x)(x + 1) = x^3 + 2x^2 + x = x^3 + x$$

Division (divisor $x + 1$, remainder 1):

$$x^3 + x + 1 = (x^2 + x)(x + 1) + 1$$

Figure 1: Example of basic operations with polynomials over the integers modulo 2

Modular arithmetic allows an efficient implementation of a form of division that is fast, easy to implement, and sufficient for the purposes of error detection. Addition and subtraction operations are equal in this arithmetic and both are the same as the XOR function over bits. There are no carries between the bits and computing of all basic operations is less computational expensive than in normal arithmetic. That is the reason why the modular arithmetic is used. The quality of generated checksum as the reminder after division are comparable in both arithmetics and it is mainly influenced by the chosen generator polynomial.

3.2 Choosing Generator Polynomial

The selection of generator polynomial is the most important part of implementing the CRC algorithm. The polynomial is chosen to maximize the error detecting capabilities (minimizing collision probability). The most important attribute of the polynomial is its length (the number of the highest nonzero coefficient), because of its direct influence of the length of the computed checksum. Commonly used polynomial lengths are 9 bits (CRC8), 17 bits (CRC16) and 33 bits (CRC32). There are also communication standards approved by IEEE or ITU organizations, which define CRC polynomials used in various communication protocols. When creating a new polynomial, general advice is to use an irreducible polynomial (over modular arithmetics), which means that the polynomial cannot be divided by any polynomial (except itself) with zero reminder. These properties of the generator polynomial can be derived from the algorithm definition below:

- CRC with more than one nonzero coefficients is able to detect all single bit errors in the input message.

- CRC can be used to detect all double bit errors in the input message shorter than 2^k, where k is the length of the longest irreducible part of the polynomial

- If the CRC polynomial is divided by $x + 1$ then no polynomial with odd number of nonzero coefficients can be divided by it. Hence, it can be used to detect odd number of errors in the input message (like single bit parity function).

- CRC polynomials detect (single) burst errors shorter than the number of the position of the highest polynomial coefficient

3.3 Table Driven CRC Implementation

Very fast and often used implementation of CRC algorithm is the table driven implementation. It is based on standard hand division algorithm where the remainder is computed by successive subtraction (XOR in modular arithmetics) of the base and the divisor. The table driven algorithm takes the input message divided into blocks. Each block value is used as the address of the relevant item in the table of precomputed CRC values. The CRC values are accumulated in order to get the final remainder - checksum [CRCGuide].

The size of the table depends on the size of input blocks. At the software level, the block size is usually 8 or 16 bits, which means 256 or 65536 table items. At the hardware level, the table can be implemented as a combinational circuit. The CRC64 in the Netflow probe processes the input with 32 bits block size.

4 Determining Properties of Generator Polynomial

Basic properties like robustness with respect to few bit errors were defined in the introduction section. But hardware developers are interested in more specialized properties. They would like to know if IP flows coming from different IP addresses or networks can produce the same CRC64 checksum which means that they are indistinguishable for the Netflow probe. The IP addresses are part of the 320 bits long packet header which is used as input for CRC64. The problem can be therefore reduced to determining the number of bits in which are two input messages producing the same checksum different (Hamming weight). We can also go further by restricting the position and bursts of the different bits.

The basic properties of the generator polynomial guarantees the minimum Hamming weight 3 for an irreducible polynomial multiplied by the *x + 1* and input messages shorter than $2^{(checksum\_length - 1)}$. Requirements for higher Hamming weights are much more complicated to deduce and are usually done by generating all possible inputs and measuring the Hamming weight [EmbCRC]. Unfortunately, it is nearly impossible to use this method for CRCs with longer generator polynomial (more than 32) due to its exponential complexity. However, this method uses very similar approach as the state space search method used by model checkers. Hence we decided to implement the CRC64 algorithm in Cadence SMV [CSMV] language and try to determine the Hamming weight by a model checker.

4.1 Version One – Table Driven Algorithm

At first, we implemented the whole table driven algorithm in the SMV language. We used the same approach to representing the table as combinational function to get off the need of precomputing it. Then we constructed formulas in CTL temporal logic saying that two different inputs cannot produce the same checksums. The length of the input message was hard-coded in the formulas. This version turned out to be useful only for CRC8 with 16bit input message. For longer input messages or checksums, the time and memory consumption ran over the usable bounds.

4.2 Version Two – XOR of Shifted Polynomials

The first version has showed that the direct CRC algorithm implementation is too complex and suffers from the state explosion problem. To overcome this problem we concentrated on the method how CRC colliding messages can be generated. CRC collisions can be easily computed by applying XOR operation on the input message and generator polynomial with various offsets. It can

be easily shown that number of colliding messages generated by this way plus number of unique messages is equal to number of all possible input messages.
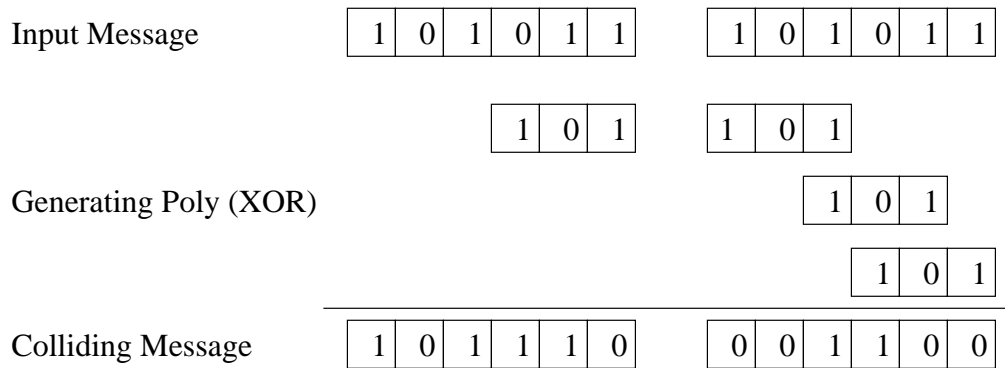

| Input Message | 1 | 0 | 1 | 0 | 1 | 1 | | 1 | 0 | 1 | 0 | 1 | 1 |

Figure 2: Example of creation of colliding messages

To obtain two input messages which are different in minimal number of bits, the generator polynomial have to be XORed with randomly shifted copies of itself, producing a string of bits with minimal number of nonzero bits. The number of nonzero bits is just the Hamming weight of the given CRC polynomial. XOR operation applied to this string and any input message produces a colliding input message with minimal Hamming weight.

We implemented this approach in SMV as a single register which is being shifted continuously to left and XORed with the generator polynomial at random time. The temporal logic formula is checking whether the number of nonzero bits in the register is greater than desired Hamming weight. The width of the input message was initially controlled by separate counter but for minimizing the number of state variables, it was lately replaced by the number of iterations performed by the model checker. Another improvement in verification performance was achieved by adding precondition about raising characteristics of the number of nonzero bits. Once if a nonzero bit is shifted out from the register, it is impossible to change its value. Therefore the number of nonzero bits cannot be decreased.

4.3   Version Two – Results

In this version, we are able to determine Hamming weight of CRC64 checksums with input message up to 128 bit length. This is not enough to verify properties in the Netflow probe which is using 320bit input messages. The limiting factor is amount of available memory and the fact that memory requirements grow exponentially to the number of input bits. The verification time is in order of tens of minutes, which is still acceptable. We tried to improve these results by using bounded model checking with satisfactory (SAT) solver, but we encountered nearly the same limits.

As a part of the research process, we focused also on other CRC algorithms with shorter checksum length. We figured out that our implementation is suitable for determining Hamming weight of CRC32 algorithm with input message up to 180 bits, where we have met the same limitations as with CRC64. We also compared determined Hamming weights for CRC16 algorithm and 48bit input messages with results presented in [EmbCRC] and got even smaller weights in some test cases.

5   Conclusion

Even if we did not verify the CRC64 properties in desired input range and further simplification of presented algorithm is not possible, we showed that verification tools can be used not only for checking safety and fairness system properties, but also for determining some special system properties like correctness of CRC checksums. However, to be able to answer desired questions about CRC64, another approach should be attempted. One option might be to improve the second algorithm with heuristics used in artificial intelligence algorithms. But this is not possible directly with the SMV model checker, so another specialized tool have to be developed to implement such a method.

References

[CompNets] Andrew S. Tanenbaum: *Computer Networks, Fourth Edition*
Prentice Hall, 2003.

[LibWWW] Liberouter: *Liberouter Project WWW Pages*
`http://www.liberouter.org/`

[StatTest]   Liberouter: *HGEN CRC64 Unit Statistical Tests*
`http://www.liberouter.org/˜hlavka/crc/crc_stats.html`

[MathWorldCRC] Eric W. Weisstein: *Cyclic Redundancy Check*
From   MathWorld   -   A   Wolfram   Web   Resource,
`http://mathworld.wolfram.com/CyclicRedundancyCheck.html`

[CRCGuide] Ross N. Williams: *A Painless Guide To CRC Error Detection Algorithms*
`http://www.ross.net/crc/crcpaper.html`

[EmbCRC] Koopman P., Chakravarty T., *Cyclic Redundancy Code (CRC) Polynomial Selection For Embedded Networks*<bri/> in proceedings of International Conference on Dependable Systems and Networks (DSN'04) 145-, 2004.

[CSMV]   Cadence SMV: *Cadence SMV WWW Pages*
          `http://www-cad.eecs.berkeley.edu/˜kenmcmil/smv/`