Investigation of the computational complexity of the formation of checksums for the Cyclic Redundancy Code algorithm depending on the width of the generating polynomial

Odilzhan A. Turdiev [a], Vladimir A. Smagin [b] and Vladimir N. Kustov [a]

[a] *Petersburg State Transport University, 9 Moskovsky pr, Saint Petersburg, 190031, Russian Federation*

[b] *International Academy of Informatization St. Petersburg, Saint Petersburg, 190031, Russian Federation*

Abstract

Formulation of the problem: The need to ensure the integrity of data transmitted in communication networks makes the issue of ensuring the formation of checksums actual. At the same time, it is advisable to reduce the complexity of algorithms for generating checksums to improve data integrity. The well-known CRC (Cyclic Redundancy Code) checksum generation algorithm has high computational complexity. **The aim of the work** is to perform search studies to substantiate the fundamental possibility of reducing the computational complexity of the algorithm for generating CRC checksums and searching for possible ways of practical implementation. The novelty of the work lies in the fact that in order to achieve the goal of the work, the computational complexity of the CRC algorithm is investigated depending on various generating polynomials and their bit width. **Result**: on the basis of the conducted search studies, the possibility of reducing the computational complexity of the algorithm for generating CRC checksums for data transmitted in communication networks was confirmed. Additionally, a set of programs is presented for assessing the complexity of the CRC checksum generation algorithm depending on the length of the polynomial, as well as the analysis of the computational complexity of the CRC-based data integrity check algorithm. **Practical relevance**: The research results can be applied to reduce the computational complexity of the algorithm for generating CRC checksums in protocols of data transmission networks, as well as to substantiate promising ways for further research in this direction.

Keywords

computational complexity, generator polynomial, cyclic redundancy code, CRC, burst errors, bit errors, data integrity.

1. Introduction

One of the important tasks in modern communication networks is to ensure data integrity. The most common algorithm for determining the integrity of transmitted data is the CRC (Cyclic Redundancy Check) algorithm.[1].

The CRC algorithm is based on the theory of cyclic error correction codes. This algorithm was first proposed by V.V. Peterson and D.T. Brown, in [1]. The CRC algorithm computes a short binary sequence that has a certain constant length, known as a check value or CRC algorithm code. The CRC is calculated for each individual block of data to be transmitted over the network and added to the block to form a codeword. When a codeword is received on the receiving side, one of two things is done. Either the received CRC check value is compared with the value of the CRC that is generated anew for the transmitted data on the receiving side, or the CRC is generated again on the receiving side for the entire received

Models and Methods for Researching Information Systems in Transport, Dec. 11-12, St. Petersburg, Russia
EMAIL: odiljan.turdiev@mail.ru (O.A. Turdiev); va_smagin@mail.ru (V.A. Smagin).
ORCID: 0000-0002-1651-5493 (O.A. Turdiev).
© 2020 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
CEUR Workshop Proceedings (CEUR-WS.org)

CEUR-WS.org/Vol-2803/paper18.pdf

codeword and the resulting CRC check value is compared with the expected residual constant. If the control values do not match, then the transmitted data contains an error and the receiving device can take actions to correct them, such as re-reading the block or sending it again.

The theoretical provisions of the functioning of the CRC algorithm are given in [2-5]. It is assumed that when a data block and its check CRC are received correctly, integrity is ensured for that data block. The process of generating and checking the CRC can be quite laborious when using network devices with low speed or high data transfer rates. In this regard, reducing the computational complexity of the CRC algorithm is an urgent scientific and practical task.

In addition, in real transmission conditions, the communication channel can be affected by various kinds of interference, which appear in the process under study in the form of erroneous bits, which lead to a violation of data integrity [6]. In the work of V.V. Yakovlev [7] proposed the choice of the generating polynomial to increase the probability of error recognition when generating checksums in the transmitted data.

Summarizing the above, it can be noted that the purpose of the work is to perform search studies to substantiate the fundamental possibility and possible ways to reduce the computational complexity of the CRC checksum generation algorithm used to control the integrity of the transmitted data.

To find ways to reduce the computational complexity in the work, a study of the computational complexity of the formation of CRC checksums was carried out depending on various generating polynomials and their bit width. To assess the computational complexity of the CRC, we simulated the process of transmitting binary data over a symmetric channel.

2. Basic concepts and characteristics of CRC codes

Cyclic redundancy codes CRC are a subclass of block codes and are used in HDLC, Token Ring, Token Bus protocols, Ethernet protocol families and other link layer protocols [8]. Computing resources mean memory, processor power, and the number of shift registers. One of the ways to represent a cyclic code is to represent it in the form of a generating polynomial - the set of all polynomials of degree (*r* – code-1), containing some fixed polynomial *G* (*x*) as a common factor. The polynomial *G* (*x*) is called the generating polynomial of the code. For example, $x^4 + x$ +1, here *r*-code = 5, since the binary sequence looks like 10011. The standardized and recommended generator polynomials for the CRC algorithm are given in Table 1, where the name of the standard and the generator polynomial are shown: for example, the notation $x^4 + x + 1$ is equivalent to $1 \cdot x^4 + 0 \cdot x^3 + 0 \cdot x^2 + 1 \cdot x + 1 \cdot x^0$ = 10011 (in binary).

Table 1.
Popular standardized generator polynomials [9].

nomination	Generating polynomials
CRC-4-TU	$^4 + x + 1$ (ITU G.704)
CRC-5-EPC	$^5 + x^3 + 1$ (Gen 2 RFID)
CRC-5-ITU	$^5 + x^4 + x^2 + 1$ (ITU G.704)
CRC-5-USB	$^5 + x^2 + 1$ (USB token packets)
CRC-6-ITU	$^6 + x + 1$ (ITU G.704)
CRC-7	$^7 + x^3 + 1$ (ITU-T G.707, ITU-T G.832, MMC, SD)
CRC-8-CCITT	$^8 + x^2 + x + 1$ (ATM HEC), ISDN Header Error Control and Cell elineation ITU-T I.432.1 (02/99)
CRC-8-Dallas/Maxim	$^8 + x^5 + x^4 + 1$ (1-Wire bus)
CRC-8	$^8 + x^7 + x^6 + x^4 + x^2 + 1$ (ETSI EN 302 307, 5.1.4)
CRC-8-SAE J1850	$x^8 + x^4 + x^3 + x^2 + 1$
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x + 1$
CRC-11	$^{11} + x^9 + x^8 + x^7 + x^2 + 1$ (FlexRay)
CRC-12	$^{12} + x^{11} + x^3 + x^2 + x + 1$ (системы телекоммуникации)

CRC-15-CAN	$x^{15}+x^{14}+x^{10}+x^{8}+x^{7}+x^{4}+x^{3}+1$
CRC-16-IBM	$^{16}+x^{15}+x^{2}+1$ (Bisync, Modbus, USB, ANSI X3.28,)
CRC-16-CCITT	$^{16}+x^{12}+x^{5}+1$ (X.25, HDLC, XMODEM, Bluetooth, SD и др.)
CRC-16-T10-DIF	$^{16}+x^{15}+x^{11}+x^{9}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+1$ (SCSI DIF)
CRC-16-DNP	$^{16}+x^{13}+x^{12}+x^{11}+x^{10}+x^{8}+x^{6}+x^{5}+x^{2}+1$ (DNP, IEC 870, 1-Bus)
CRC-24	$^{24}+x^{22}+x^{20}+x^{19}+x^{18}+x^{16}+x^{14}+x^{13}+x^{11}+x^{10}+x^{8}+$ $^{6}+x^{3}+x+1$ (FlexRay)
CRC-24-Radix-64	$^{24}+x^{23}+x^{18}+x^{17}+x^{14}+x^{11}+x^{10}+x^{7}+x^{6}+x^{5}+x^{4}+$ $^{3}+x+1$ (OpenPGP)
CRC-30	$^{30}+x^{29}+x^{21}+x^{15}+x^{13}+x^{12}+x^{11}+x^{8}+x^{7}+x^{6}+x^{2}+x+$ (CDMA)
CRC-32-IEEE 802.3	$^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+$ $^{2}+x+1$ (V.42, MPEG-2, PNG, POSIX cksum)
CRC-32C (Castagnoli)	$^{32}+x^{28}+x^{27}+x^{26}+x^{25}+x^{23}+x^{22}+x^{20}+x^{19}+x^{18}+x^{14}+$ $^{13}+x^{11}+x^{10}+x^{9}+x^{8}+x^{6}+1$
CRC-32K (Koopman)	$^{32}+x^{28}+x^{27}+x^{26}+x^{25}+x^{23}+x^{22}+x^{20}+x^{19}+x^{18}+x^{14}+$ $^{13}+x^{11}+x^{10}+x^{9}+x^{8}+x^{6}+1$
CRC-32Q	$^{32}+x^{31}+x^{24}+x^{22}+x^{16}+x^{14}+x^{8}+x^{7}+x^{5}+x^{3}+x+1$
CRC-64-ISO	$^{64}+x^{4}+x^{3}+x+1$ (HDLC — ISO 3309)
CRC-64-ECMA	$^{64}+x^{62}+x^{57}+x^{55}+x^{54}+x^{53}+x^{52}+x^{47}+x^{46}+x^{45}+x^{40}+$ $^{39}+x^{38}+x^{37}+x^{35}+x^{33}+x^{32}+x^{31}+x^{29}+x^{27}+x^{24}+x^{23}+$ $^{22}+x^{21}+x^{19}+x^{17}+x^{13}+x^{12}+x^{10}+x^{9}+x^{7}+x^{4}+x+1$

The value of the result of the implementation of the CRC algorithm is the remainder of dividing the binary sequence *M* (*x*) corresponding to the input data by the generating binary sequence *G* (*x*) of degree r.

Each final sequence of bits is one-to-one associated with a binary polynomial, the sequence of which is the original sequence. For example, bit sequence 1011010 corresponds to the binary polynomial $M(x) = 1\cdot x^6 + 0\cdot x^5 + 1\cdot x^4 + 1\cdot x^3 + 0\cdot x^2 + 1\cdot x^1 + 0\cdot x^0 = x^6 + x^4 + x^3 + x$.

Let's consider examples in which addition and subtraction are performed without carrying the digits in accordance with the "exclusive OR" operation, which corresponds to the addition modulo 2 of binary arithmetic.

```
  10011011      00110011      11110000       01010101
 +11001010     + 11001101    + 10100110     + 10101111
 ---------     ---------     ---------      ---------
  01010001      11111110      01010110       11111010
```

Polynomial division is performed in the binary system, with the difference that the subtraction is performed modulo 2.

The use of polynomial codes during transmission is as follows. The sender and the receiver pre-select the same generator polynomial *G (x*), whose coefficients at the higher term and at the lower term must be equal to 1. When calculating the checksums of a block of m bits in size, the following condition $m > r$ must be met. Further, implementing the CRC calculation algorithm, the sender adds the checksum to the transmitted block, considered as a polynomial *M (x),* so that the transmitted block with the checksum is a multiple of *G (x).* When the recipient receives the checksum block, he divides it by *G (x).* If a nonzero remainder is formed, then this indicates the occurrence of an error during transmission [10,11].

Checksum calculation algorithm:

1. Add *r* zeros to the end of the block so that it contains *m* + *r* digits, the result is a polynomial $x^rM(x)$.
2. Divide the polynomial $x^rM(x)$ by *G (x)* modulo 2, the quotient is ignored.
3. Subtract modulo 2 the remainder (its length does not exceed *r* bits) from the string corresponding to $x^rM(x)$. The result is a block with a checksum.

Currently, most network technologies most often use the following three types of generating polynomials *G (x)* [12]:

CRC-12 $= x^{12} + x^{11} + x^3 + x^2 + x + 1$

CRC-16-IBM $= x^{16} + x^{15} + x^2 + 1$

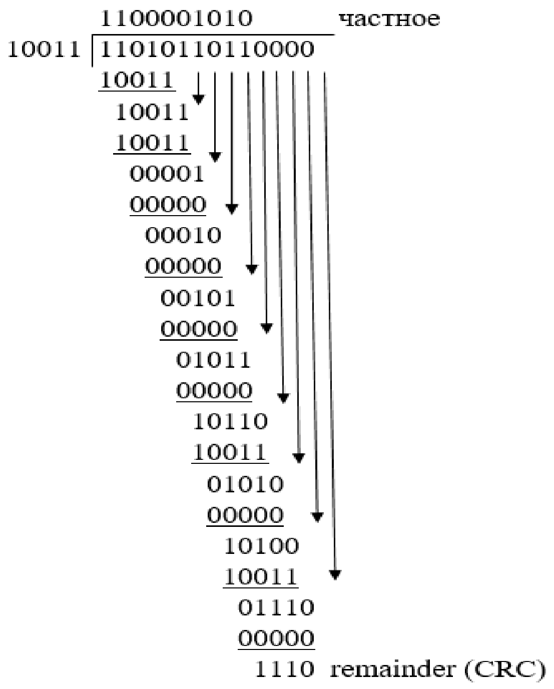
CRC-16-CCITT $= x^{16} + x^{12} + x^5 + 1$

CRC-12 is used to transmit 6-bit characters. The other two are for 8-bit. CRC-16 and CRC-CCITT detect all single and all double errors, an odd number of isolated errors, single bursts of errors less than 16 in length, and many burst errors greater than 16 with a 99.997% probability.

After getting acquainted with the basic concepts and characteristics of CRC codes, let us consider an example of calculating the remainder of cyclic redundancy codes and estimate the computational complexity of this process.

3. An example of calculating the remainder for constructing a CRC code word and estimating the computational complexity

Calculation of CRC.
Information frame - 1101011011
Generator polynomial - 10011
Frame with additional zeros – 11010110110000



CRC codeword (transmitted frame) - 11010110111110 [13-15].

Thus, the example shows that with the generator polynomial CRC-4-TU (10011) and the bit length of the information frame equal to 10 bits, 10 divisions and 50 modulo 2 additions are required. In the general case, the relationship between the computational complexity (the number of additions and divisions), the size of the generating polynomial and the size of the information polynomial.

For the best understanding of the number of divisions and additions in the CRC algorithm with various generator polynomials and information frames, software has been created to assess the computational complexity of the CRC algorithm.

4. A program for evaluating the computational complexity of the CRC algorithm

The program is designed to estimate the complexity of the CRC algorithm by the width of the polynomial using the CRC checksum method, which is shown in the example of **cyclic redundancy codes**. By estimating the complexity of the implementation of the CRC algorithm by performing additions according to the width of the polynomial, we obtain an approximate number of additions (shift into cells). This study opens up possible ways to reduce the computational complexity of the CRC checksum generation algorithm used to control the integrity of transmitted data [16,17].

For the considered CRC generation method, the "CRC calculation" application was written in Java, presented in Appendix 1, which allows for a user-specified frame size and generating polynomial to estimate the required number of additions and divisions to obtain the final CRC checksum. The javaFX library was used to write the application. The CRC and TableData classes are used from this library. There are three operations in the CRC class:

1. Operation (calculation) - it calculates the CRC code.
2. Operation (initialize).
3. Operation (xor).

The initialize operation processes the bit sequence array and passes it to the table, the xor operation emulates the operations of logical addition of the bits of the input code and the CRC code to the input and gives two parameters X and Y - a binary code, 1 and 1, or 1 and 0, or 0 and 1, or 0 and 0.

5. Analysis of the results of evaluating the computational complexity of the CRC algorithm

The calculation of the number of additions, which is one of the important components of the CRC algorithm, is carried out in accordance with formula 1. Note that the value obtained as a result of the calculations should not be small, as this will lead to the need to retransmit the data. In this case, the transmission algorithm is repeated, which leads to secondary calculations. Polynomials are chosen so that there is no downtime in the data transmission channels.

The calculation of the number of additions is carried out according to the formula:

$$R = (P \cdot N)\ mod\ 2, \qquad (1)$$

here R is the number of addition operations modulo 2;

P - packet size;

N - is the length of the polynomial.

Let us consider an example in which the digit capacity of 8 polynomials has 48 divisions, therefore the number of additions according to the formula (1) $R = 48 \cdot 8 = 384$. The results of evaluating the computational complexity of the CRC generation algorithm, expressed in the number of operations, are shown using the graph in Fig. 1 and are confirmed a simulation model.

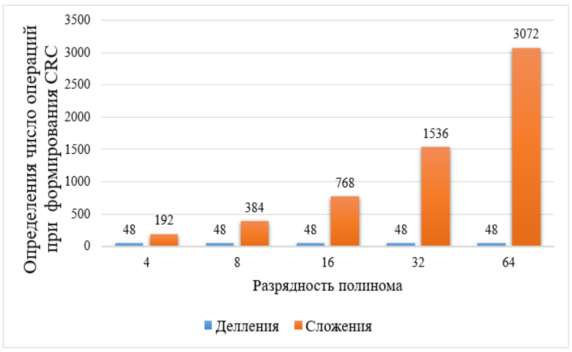


Figure 1: The result of evaluating the computational complexity of the CRC generation algorithm, expressed in the number of operations.

Conclusions

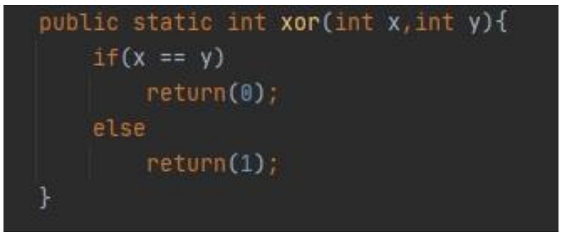
The article investigates the computational CRC algorithm. It is shown that with an increase in the number of bits of generating polynomials, the number of operations increases, while the number of division operations remains unchanged. With an increase in the number of digits and division operations, the generating polynomials remain unchanged, but the number of additions increases.

It is shown that a possible way to reduce the computational complexity of the CRC checksum generation algorithm used to control the integrity of the transmitted data is to reduce the number of addition operations while maintaining the number of division operations in the algorithm under consideration.

As a result of this work, on the basis of research on generating polynomials and the number of division operations, an expression for calculating the number of addition operations was obtained. The research results pave the way for further research to reduce the computational complexity of the operation of cyclic redundancy codes.

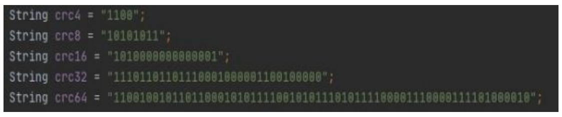
Appendix 1.

By the logic of the method, the logical operator "or" is emulated, that is, xor, as shown in Listing 1.


```
public static int xor(int x,int y){
    if(x == y)
        return(0);
    else
        return(1);
}
```

Listing 1: Emulation of the logical operator "or".

The CRC class contains fields with the original message, that is, a variable that stores a bit sequence of a certain length, as shown in Listing 2.



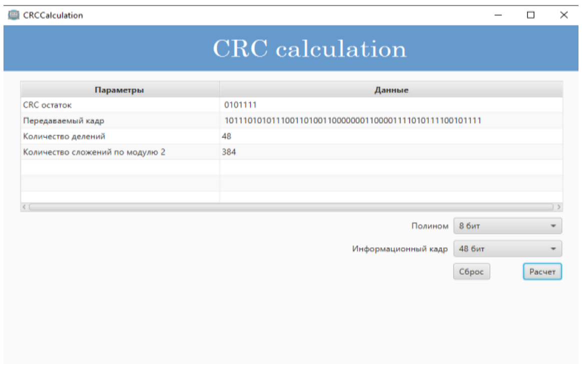
```
String crc4 = "1100";
String crc8 = "10101011";
String crc16 = "1010000000000001";
String crc32 = "11101101101110001000001100100000";
String crc64 = "1100100101101100010101111001010111010111100001110000111101000010";
```

Listing 2: Variable storing a bit sequence of a certain length.

Next, a polynomial of a certain bit depth is generated and the CRC code is calculated in the calculation method.

The program simulates the process of data transfer between the source and the receiver (Fig. 2), and also provides the following functions:

1. Specifying the initial data in the specified table ("polynomial" and "information frame", Fig. 2) creates a number generator using a unique seed.
2. Calculation of the initial data is displayed in the program window (button "Calculation", Fig. 2).
3. As a result of the initial data, the number of additions and the formation of CRC checksums of cyclic redundancy codes (“parameters and data” field, Fig. 2) is calculated.
4. Additionally, at the discretion of the user, you can reset and start the calculation over again (the "Reset" button, Fig. 2).



Параметры	Данные
CRC остаток	0101111
Передаваемый кадр	1011101010111001101001100000001100001111010111100101111
Количество делений	48
Количество сложений по модулю 2	384

Полином: 8 бит
Информационный кадр: 48 бит

Figure 2: CRC calculation.

After the simulation of the data calculation process, the number of addition of the CRC checksum algorithm of cyclic redundancy codes is calculated again, which are then compared with those previously calculated to reveal the fact of calculating the addition of the checksum algorithm of the transmitted data (Fig. 2); field "CRC calculation".

References

[1] Peterson, W. W. and Brown, D. T. (January 1961). "Cyclic Codes for Error Detection". Proceedings of the IRE 49: 228.

[2] Ritter, Terry (February 1986). "The Great CRC Mystery". Dr. Dobb's Journal 11 (2): http://www.ciphersbyritter.com/ARTS/CRCMYST.HTM. Retrieved 21 May 2009 26–34, 76–83.

[3] N. Cam-Winget, Nancy; R. Housley, Russ; D. Wagner, David; J. Walker, Jesse (May 2003). "Security Flaws in 802.11 Data Link Protocols". Communications of the ACM 46 (5): 35–39.

[4] Stigge, Martin; Plötz, Henryk; Müller, Wolf; Redlich, Jens-Peter (May 2006). Reversing CRC – Theory and Practice (http://sar.informatik.hu-berlin.de/research/publications/SAR-PR-2006-05/SAR-PR-2006-05_.pdf). Berlin: Humboldt University Berlin. p. 17. Retrieved 4 February 2011

[5] Anachriz (30 April 1999). "CRC and how to Reverse it" Retrieved 21 January 2010. Online essay with example x86 assembly code.

[6] "Eurocontrol – FAQ: Technologies" (http://www.eurocontrol.int/aim/public/faq/chain faq3.html). European Organisation for the Safety of Air Navigation. 29 April 2009.

[7] Yakovlev V.V., Fedorov R.F. Stochastic computers. L., "Mechanical Engineering", 1974,

[8] Ross N. Williams Элементарное руководство по CRC – алгоритмам обнаружения ошибок. Copyright (C) Ross Williams, 1993

[9] Cyclic redundant code [Electronic resource]. - Internet page. - Access mode: https://ru.wikipedia.org/wiki/Cyclic_Redundant_code, free

[10] Yakovlev VV Assessment of the influence of interference on the performance of the channel level protocols / VV Yakovlev, FI Kushnazarov // Izv. Petersburg. state un-that ways of communication. - SPb .:

PGUPS, 2015. - Issue. 1 (42). - S. 133-138.

[11] Halsall F. Fifth edition, computer networks and the Internet / F. Halsall. – Addison-Wesley: Pearson Education, 2005. – 803 p.

[12] Lin S. and Costello D.J. Jr. Error Control Coding: Fundamentals and Applications. Prentice-Hall, Inc., EnglewoodCliffs, N. J., 1983.

[13] Halsall F. Data communications, computer networks and open systems / F. Halsall. – Addison-Wesley: Pearson Education, 1996. – 907 p.

[14] Olifer V. G. Computer networks. Principles, technologies, protocols / V. G. Olifer, N. A. Olifer. - SPb .: Peter, 2008 .-- 958 p.

[15] A. Romashchenko, A. Rumyantsev, A. Shen. NOTES ON THE THEORY OF CODING. Notes on coding theory. | 2nd ed., Rev. and add. | M .: MTsNMO, 2017. | 88 p. ISBN 978-5-4439-0689-8

[16] Turdiev O.A., Yakovlev V.V., Klimenko S.V., Boltaev A.Kh. Investigation of the formation of the block checksum (BCC) of the transmitted data. Izvestia ETU "LETI" Issue No. 6 2019.

[17] Turdiev O.A., Klimenko S.V., Tukhtakhodzhaev A.B. Evaluations of the efficiency of detecting checksum errors (CRC) of transmitted data Izvestia ETU "LETI" Issue No. 8 2019.