The Study of Implementations of CRCs Algorithms

Sergey V. Klimenko, Valentin V. Yakovlev
Department of Information Systems and Technologies, Emperor Alexander I St. Petersburg State Transport University
Saint Petersburg, Russia
s.klimenko@live.ru, jakovlev@pgups.ru

Boris V. Sokolov
Laboratory of Information Technologies in System Analysis and Modeling,
St. Petersburg Institute for Informatics and Automation of the RAS,
Saint Petersburg, Russia
sokolov_boris@inbox.ru

Abstract

Objective: To give comparative assessment of the basic ways of generating a checksum (CRC code) based on direct, table and matrix algorithms. **Methods:** Algorithms were compared by means of mathematical methods. In order to achieve the result Java Development Kit software version 1.8 and NetBeans IDE 8.2 development environment were used. **Results:** The methods of generating checksums by means of algorithms were described in detail. For each method under consideration, the time characteristics of their work were given. The comparison of the analyzed methods was conducted. **Practical importance:** Based on the results of the experiment, it was concluded which method was optimal for the generation of checksums.

1 Introduction

Error detection methods are intended to detect distortions in messages when they are transmitted through noisy channels. For this, the transmitter calculates a number, called a checksum, which is a function of the message, and adds it to this message. Receiver, using the same algorithm, calculates the checksum of the received message and compares it with the transmitted value [1].

CRC (Cyclic Redundancy Check) checksum is a value calculated from a data set using mathematical algorithms that provide hash-collision resistance [2]. Hash-collision is a checksum equality for various input blocks of data. Checksums are widely used to control the correctness of stored and transmitted information.

The CRC was invented by W. Wesley Peterson in 1961; the 32-bit CRC function, used in Ethernet and many other standards, is the work of several researchers and was published in 1975.

The logical prerequisite for using this type of checksum is that the size of the checksum is much smaller than the format of the converted numbers / messages, therefore, the probability of distorting the checksum (when transmitting information through any channel or when it is stored on a data storage device) is significantly lower than the probability of distorting an array of information.

CRCs are so called because the check (data verification) value is a redundancy (it expands the message without adding information) and the algorithm is based on cyclic codes. CRCs are popular because they are simple to implement in binary hardware, easy to analyze mathematically, and particularly good at detecting common errors caused by noise in transmission channels. Because the check value has a fixed length, the function that generates it is occasionally used as a hash function.

The basic idea of the CRC algorithm is to present the message as a huge binary number, divide it by another fixed binary number, and use the rest of this division as a checksum. Having received the message, the recipient can perform a similar action and compare the balance received with the "checksum".

The simplest error-detection system, the parity bit, is in fact a 1-bit CRC: it uses the generator polynomial x + 1 (two terms), and has the name CRC-1.

Cyclic redundant codes (CRC) are a subclass of block codes and are used in HDLC, Token Ring, Token
Bus protocols, Ethernet protocol families and in other protocols of a link level. The popularity of CRC codes is due to the fact that the procedures encoding and decoding are fairly simple and do not require large computational resources.

In: S. V. Klimenko, V. V. Yakovlev (eds.): Selected Papers of the Workshop Computer Science and Engineering in the framework of the 5th International Scientific-Methodical Conference "Problems of Mathematical and Natural-Scientific Training in Engineering Education", St.-Petersburg, Russia, 8–9 November, 2018, published at http://ceur-ws.org

The CRC algorithm is based on the properties of division with the remainder of binary polynomials, thus, the CRC value is the remainder from dividing the polynomial corresponding to the input data by some fixed generating polynomial [1].

The most important task of constructing CRC codes is the choice of the generating polynomial. There are many standardized and recommended by various organizations generating polynomials used to generate CRC. For example, CRC32 generating polynomial of IEEE 802.3 standard looks like this:

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1 =$$
$$0x04C11DB7(\text{in hexadecimal form}) =$$
$$100110000010001110110110111$$
$$(\text{in binary form}).$$

One of the main conditions for choosing a polynomial is that the coefficients in the high and low bits were equal to one.

The mathematical model of finding the checksum is presented below:

$$R(x) = P(x) \cdot x^N \bmod G(x), \quad (1)$$

where R(x) – polynomial which represents the value of CRC; P(x) – polynomial, in which coefficients represent input blocks of data; G(x) – generating polynomial; N – degree o generating polynomial (1≤ N ≤ 256).

Thus, the CRC calculation is possible to implement on the basis of any programming language by using XOR (Exclusive or) and SHL ("shift to the left") logical operations, because they are included into any programming language [3, 4].

An example of the execution of the algorithm for calculating the CRC remainder is shown in Fig. 1, where the polynomial 10011 is selected as the generator.

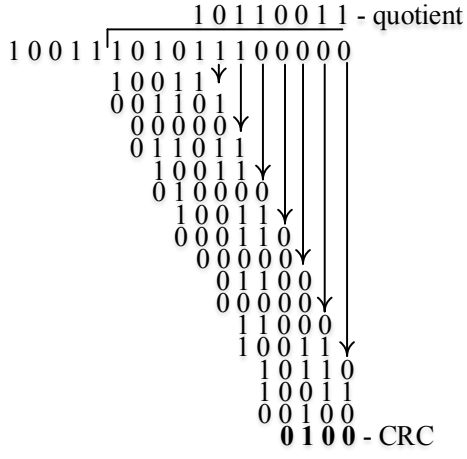

Figure 1: Example for CRC calculation

The probability that the distortion of a transmitted message in several positions will be such that the final checksum does not change is determined by the formula [5–8]:

$$P = \frac{1}{2^N} \quad (2)$$

CRCs are specifically designed to protect against common types of errors on communication channels, where they can provide quick and reasonable assurance of the integrity of messages delivered. However, they are not suitable for protecting against intentional alteration of data.

Firstly, as there is no authentication, an attacker can edit a message and recompute the CRC without the substitution being detected. When stored alongside the data, CRCs and cryptographic hash functions by themselves do not protect against intentional modification of data. Any application that requires protection against such attacks must use cryptographic authentication mechanisms, such as message authentication codes or digital signatures (which are commonly based on cryptographic hash functions).

Secondly, unlike cryptographic hash functions, CRC is an easily reversible function, which makes it unsuitable for use in digital signatures.

Thirdly, CRC is a linear function with a property that:

$$crc(x \oplus y \oplus z) = crc(x) \oplus crc(y) \oplus crc(z) \quad (3)$$

2 Brief description of CRC counting algorithms

The direct CRC calculation algorithm determines the control CRC bit by bit [9], it is described as follows in accordance with (1):

1) add zeros to the original message for alignment (the number of zeros is determined by the degree of the generating polynomial)$P(x)' = P(x)000 \dots N$;

2) do SHL operation of the bit sequence of the message P (x) ′ until the bit in the cell becomes equal to one or the number of bits becomes less than in the divider;

3) if high bit becomes equal to one, then perform an XOR operation between the message and the generating polynomial and repeat step 2;

4) the final residue of the sequence P (x)′ is a CRC-residue.

In the above description, G (x) is a polynomial, N is the degree of the polynomial, P (x) is the original message, and P (x) ′ is the augmented original message.

The need to perform multiple iterations when generating the CRC checksum results in significant time costs.

The tabular CRC calculation algorithm is used to accelerate the calculation of the CRC checksum.

Acceleration is accomplished by replacing eight shift operations with a single search operation in the table, which contains 256 values. Therefore, when calculating the CRC checksum, a cycle is performed on 256 values.

The prerequisite for the appearance of the table was the fact that when performing a XOR operation with a constant value at different shifts, there will always be some value, which, when applying an

XOR operation with the original content, will give the same result. Therefore, it is possible to build a table of such values, where index is the original content [1].
Table building algorithm:
1) calculate the value in the table for each byte from 0x00 to 0xff:
a) perform the "right shift" operation 8 times, and if the low bit is equal to one, then the XOR operation is performed with the polynomial G;
b) all that remains of two bytes becomes the value in the table.

Calculating algorithm the CRC checksum using the table:
1) each byte of the P (x) message is viewed:
a) an XOR operation is performed on the low byte of the current CRC value and the current byte of the message — this is the index in the table;
b) the high byte of the current CRC value shifts to the right by 8 and becomes low, then merged by XOR with the value of the table - this will be the new CRC value;
2) the result is the CRC value.

The matrix CRC calculation algorithm is used to calculate the checksum and is similar to the tabular one, except for that instead of a table, the vector multiplication operation (extended byte) by the matrix is used.

The main advantage of the matrix algorithm over the tabular is the size of the memory required to store the table. So, for the implementation of the tabular algorithm, 1 Kb (256 elements of 4 bytes each) of the memory is required to store the table, while for the matrix algorithm, only 32 bytes are required (8 elements of 4 bytes each) [10].

3 Evaluation of temporal effectiveness

In order to compare the considered methods of generating CRC by performance under the same conditions, an application was written in the high-level language Java, which allows to get statistics for each of the methods with the generating polynomial 0xEDB88320 for CRC32 and 0xd800000000000000 for CRC64. Statistics show the dependence of their execution time on the size of the source line (message).

The Java Development Kit version 1.8 and the NetBeans IDE 8.2 development environment are used to write the application. The experiment was conducted using the following hardware and software resources:
1) Windows 10 operating system (64-bit);
2) dual-core processor Intel Core i7-7600U with a clock frequency of 2.8 GHz;
3) 16 GB of RAM;
4) SSD with 512 Gb capacity.

The application generates strings (messages) in a randomly specified size based on the characters: "A"–"Z", "a"–"z", "0"–"9", and punctuation marks, after that a checksum is constructed for the obtained string using the methods described earlier. For example, a randomly generated string consisting of 12 characters, looks like this:

$$O\ h\ 3Tj\ +\ J71L.$$

To determine the running time of the algorithms, the nanoTime () static method of the System class of the java.lang package was used, which returns the current time value.
Thus, it is possible to calculate the running time of the entire application or its separate fragment (it is shown at the Figure 2). In considered case, were used the possibility of this method and the measurement of the operating time of a separate fragment (method) of the application, determined by the type of algorithm.

The analysis was performed for randomly generated rows of the following sizes (h, in MB):
1, 2, 4, 8, 16, 32, 64 and 128.

To calculate the arithmetic average of the checksum calculation time, the following formula was used:

$$\bar{t} = \frac{1}{P} \cdot \sum_{i=1}^{P} t_i, \quad (4)$$

t_i is the time for calculating the *i* checksum, P – total number.
Standard deviation was calculated as follows:

$$\sigma = \sqrt{\frac{1}{P} \cdot \sum_{i=1}^{P} (t_i - \bar{t})}, \quad (5)$$

where t is the average value.

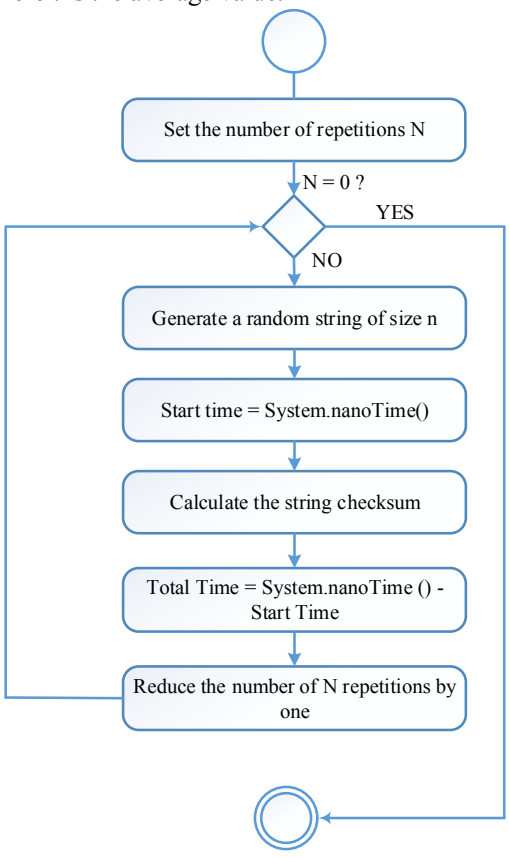

Figure 2: Flowchart - test structure

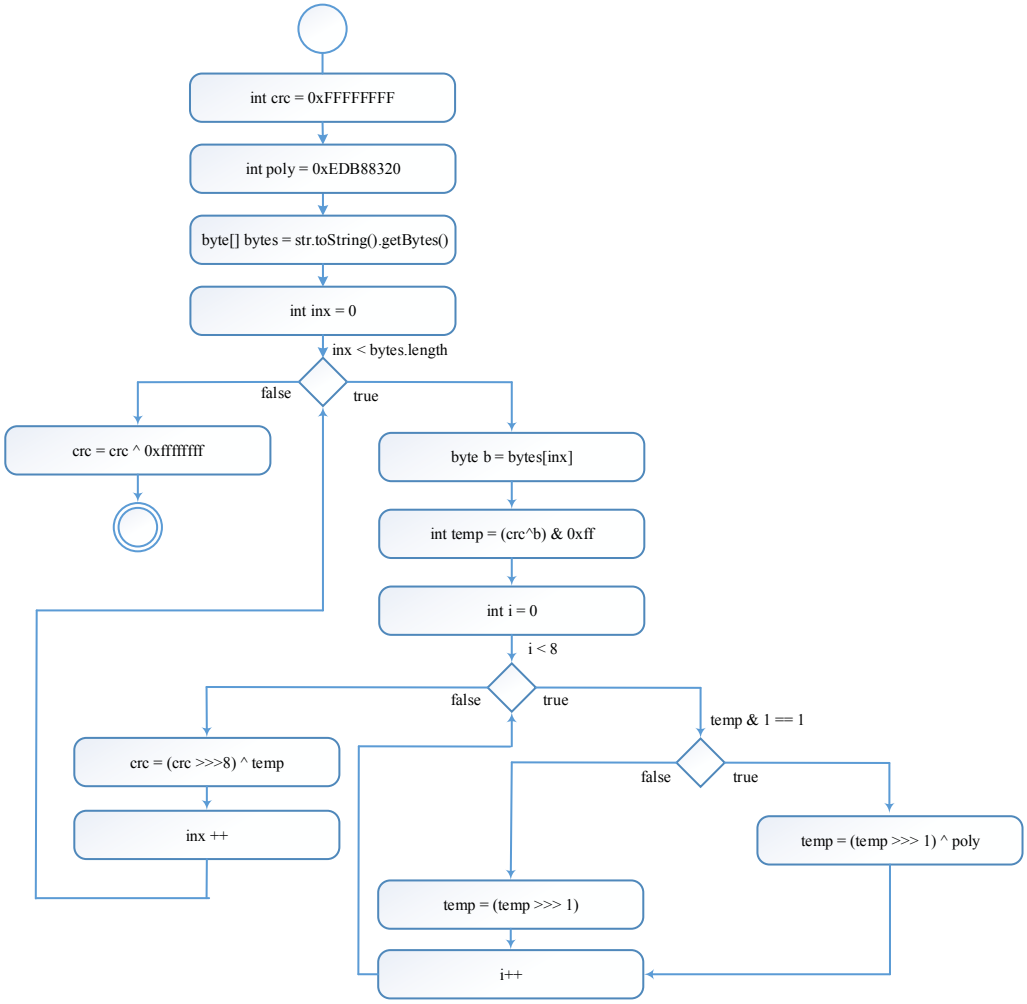

Figure 3: Checksum calculation

4 Results

The results of calculations by formulas (4) - (5) are shown in the table 1 (CRC32) and table 2(CRC64).

Table 1: Dependence of the working time of CRC32 calculation algorithms on the size of the string

h, Mb	Tabular algorithm		Direct algorithm		Matrix algorithm	
	$\bar{t}$, s	σ, s	$\bar{t}$, s	σ, s	$\bar{t}$, s	σ, s
1	0.0024	0.0008	0.0085	0.0019	0.0065	0.0013
2	0.0046	0.0007	0.0165	0.0021	0.0128	0.0017
4	0.009	0.001	0.035	0.005	0.027	0.004
8	0.019	0.005	0.07	0.013	0.053	0.01
16	0.036	0.004	0.130	0.016	0.101	0.012
32	0.072	0.003	0.256	0.014	0.199	0.01
64	0.14	0.01	0.52	0.02	0.40	0.02
128	0.29	0.02	1.03	0.07	0.80	0.04

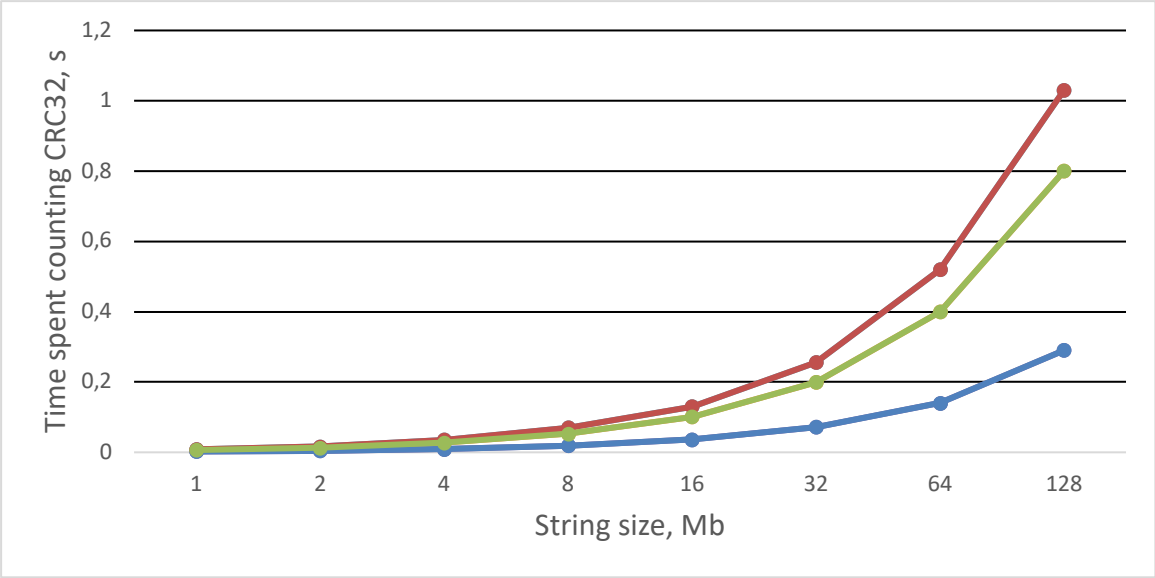

Figure 4: Dependence of the working time of direct (red), matrix (green) and tabular (blue) CRC32 calculation algorithms on the size of the string

Table 2: Dependence of the working time of CRC64 calculation algorithms on the size of the string

h, Mb	Tabular algorithm		Direct algorithm	
	$\bar{t}$, s	σ, s	$\bar{t}$, s	σ, s
1	0.005	0.001	0.009	0.001
2	0.010	0.002	0.019	0.003
4	0.018	0.003	0.036	0.003
8	0.034	0.004	0.070	0.003
16	0.067	0.008	0.144	0.009
32	0.134	0.028	0.287	0.050
64	0.445	0.116	0.557	0.020
128	0.576	0.082	1.173	0.143
256	1.218	0.064	2.241	0.063

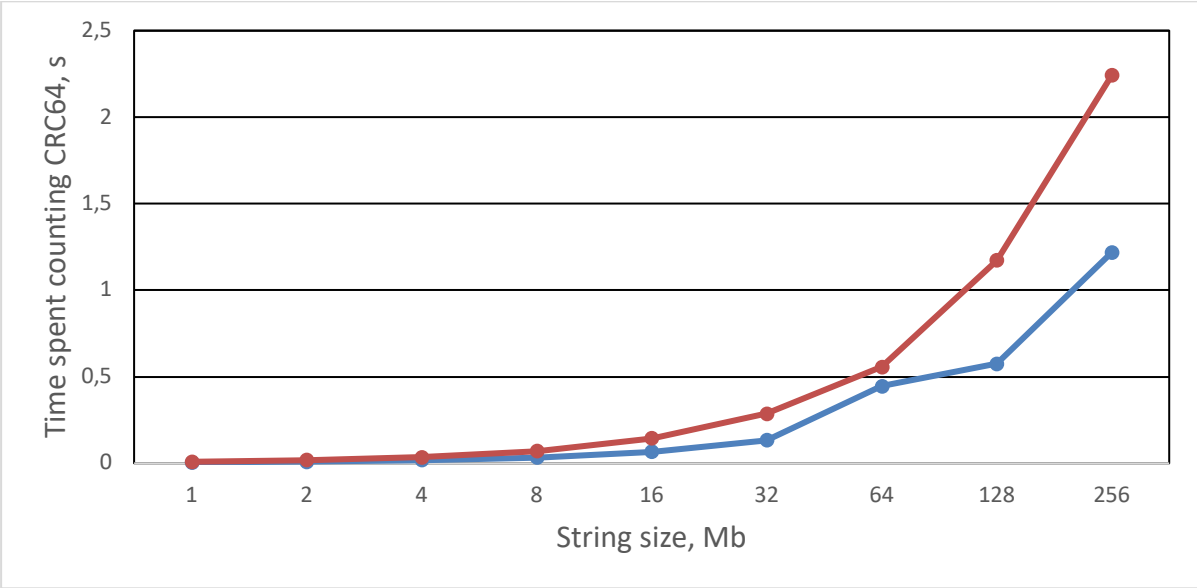

Figure 5: Dependence of the working time of direct (red) and tabular (blue) CRC64 calculation algorithms on the size of the string

As can be seen from the results given in Tables 1-2, the time spent on the calculation of the checksum CRC32 and CRC64 by the direct algorithm is almost the same. But, the calculation of CRC32 in tabular way gives a gain of 2 two times compared with CRC64.

In turn, it is worth noting that in CRC64 the probability of not detecting an error is significantly lower than $\frac{1}{2^{64}}$ versus $\frac{1}{2^{32}}$.

5 Conclusion

The article describes the direct, tabular and matrix algorithms for calculating the checksum CRC32 and CRC64(tabular and direct). A comparison of the operating time is made (Fig. 3, Fig. 4).

As a result, it was established that the tabular algorithm is optimal in terms of the calculation speed of the checksum CRC32. Acceleration in it compared to the direct algorithm was achieved by replacing eight shift operations with a single search operation in the table, which contains 256 values. It is also important that the use of the matrix algorithm can be conditioned to significant memory savings compared to the table algorithm.

Acknowledgments

Research carried out on this topic was carried out with partial financial support from RFBR grants (No. 17-29-07073-ofi-m, 18-07-01272, 19–08–00989), under the budget theme 0004.

References

[Ros93] N. W. Ross A Painless guide to CRC error detection algorithms. Australia, 1993. URL:http://www.ross.net/crc/download/crc_v3.txt (accessed: 01.04.2018).

[Myt11] E. A. Mytsko, A. N. Malchukov Osobennosti programmnoy realizatsii vychisleniya kontrolnoy summy CRC32 na primere PKZIP, WINZIP, ETHERNET [The specifi cities of software implementation of CRC32 checksum calculation by the example of PKZIP, WINZIP, ETHERNET]. Vestnik nauki Sibiri [The Siberian Science newsletter], 2011, no.1 (1), pp. 279–282. (In Russian)

[Mal10] A. N. Malchukov, A. N. Osokin Bystrodeistvuyushchiye algoritmy vychisleniya kontrolnoy summy na primere CRC8 [Fast algorithms for calculating the checksum using the example CRC8]. Molodezh I sovremennyye informatsionnyye tekhnologii: Sbornik trudov VIII Vseros. nauchno-prakt. konf. studentov, aspirantov i molodykh uchenykh [The youth and modern information technologies: Proceedings of the 8th All-Russian research and training conference of students, postgraduates and young scientists]. Tomsk, March 3–5th, 2010. Tomsk, SPB Grafi ks, 2010, pp. 34–35. (In Russian)

[Bur06] Yu. B. Burkatovskaya, A. N. Malchukov, A. N Osokin. Bystrodeystvuyushchiye algoritmy deleniya polinomov v arifmetike po modulyu dva [Fast-acting algorithms for dividing polynomials in arithmetics modulo two]. Izvestiya Tomskogo politekhnicheskogo universiteta [Proceedings of Tomsk Polytechnic University], 2006, no. 1 (309), pp. 19– 24. (In Russian)

[Koo02] P. Koopman 32-Bit Cyclic Redundancy Codes for Internet Applications. Intern. Conf. on Dependable Systems and Networks (DSN) (Washington, July 2002). Washington, DC, 2002, pp. 459–468.

[Jak15] V. V. Jakovlev, F. I. Kushnazarov Otsenka vlijanija pomekh na proizvoditelnost protokolov kanalnogo urovnya [Estimating the impact of interference on the performance of link layer protocols]. Izvestija Peterburgskogo universiteta putej soobshchenija [Proceedings of Petersburg State Transport University], 2015, issue 1 (42), pp. 133–138. (In Russian)

[Hal96] F. Halsall Data communications, computer networks and open systems. Addison-Wesley, Pearson Education Press., 1996, 907 p.

[Oli08] V. G. Olifer, N. A. Olifer Kompjuterniye sety. Printsipy, tekhnologii, protokoly [Computer networks. Principles, technologies, protocols]. Saint Petersburg, Peter Publ., 2008, 958 p. (In Russian)

[Tem79] F. E. Temnikov, V. A. Afonin, V. I. Dmitriev Teoreticheskie osnovy informatsionnoy tekhniky: uchebnoye posobiye [Theoretical basis of information technology: study guide]. Moscow, Energia Publ, 1979, 512 p. (In Russian)

[Mal10] A. N. Malchukov, A. N. Osokin Bystroje vychislenije kontrolnoy summy CRC: tablitsa protiv matritsy [Fast calculation of CRC checksum: table versus matrix]. Prikladnaja informatika [Applied Informatics], 2010, no. 2 (26), pp. 58–63. (In Russian)