Method of Adaptive Error Control Coding of Structured Messages Using Parameterization of Reed-Solomon Codes

Roman Andrushchenko and Andrii Rohovenko

Abstract More and more business processes are going online, and special information services, APIs, SDKs are created that interact with people and other information systems. In the course of their operation, the TCP/IP network stack protocols are mostly used for communication. The wireless communication devices, mobile devices: tablets, smartphones, etc. are also becoming increasingly popular. All this leads to the need to pay more attention to the quality and security of data transmission over wireless networks, as well as to improve methods of detecting and correcting of errors that occur during data reception and transmission. The article describes the method of error correction in data packets based on the structure analysis of messages, analyses possible ways of implementation and analyses its effectiveness in combination with Reed-Solomon codes. The proposed method will improve data transmission processes over network protocols.

Keywords HTTP · JSON · Data serialization · Error control coding · Reed-Solomon codes · Computer network · Network software

1 Introduction

1.1 Target Settings

Information systems have been increasingly used in various activity spheres in recent decades, and in the last few years, the special attention has been paid to wireless communications and the Internet of Things [1], which require special attention to the reliability and security of data channels used by nodes of information systems for interaction and communication with each other and with other information systems. The number of devices connected to the Internet is growing at an incredible rate,

R. Andrushchenko (✉) · A. Rohovenko
Chernihiv Polytechnic National University, 95 Shevchenko Str., Chernihiv 14035, Ukraine
e-mail: arbamor@ukr.net

A. Rohovenko
e-mail: arogovenko@gmail.com

which is the main reason for the development of IPv6 protocol in communications due to the need to expand the address space and ensure growth. A lot of Internet of Things devices connected to the global network appeared in a last few years. The combination of these factors results in the need to research, develop and improve the data transmission methods in telecommunication systems [2, 3].

The error detection and correction process is typically implemented as a part of low-level protocols that consider transmitted data as a stream or blocks of bits/bytes, without taking into account the high-level structure and semantics of messages [4, 5].

So, some of the piece of information that could theoretically be used for more successful data decoding is ignored. This approach is generally proven and mostly correct, as the protocols are low-level and should not consider the data transmitted. They operate with a set of bits only. However, all these protocols do not always cope with the tasks assigned to them, especially in the case of interference, noise that occurs during the transmission process. Therefore, it is advisable, if possible, to use all appropriate ways to improve the process of detecting and correcting of errors, including the use of as much information available as possible for recovery. One way to improve the process is to use knowledge about the data and messages transmitted, which are considered not just as a set of bits, but as a structure with its own characteristics.

The TCP/IP protocol stack is the most common in computer networks. It consists of several layers, and at each level, each protocol of the same layer is built on a protocol of a lower layer and each protocol has its own specific task. The lower layer protocols (which belong to the physical/channel layers of the OSI model) deal with, among other things, the detection and correction of errors that may occur during data transmission. These protocols consider data as a stream of bits/bytes and do not take their semantics, structure and content into account. In this regard, some of the information that could be used to improve the reliability of data transmission is not utilized. On the one hand, this approach simplifies the implementation of communication channels, but on the other hand, in case of interference in data transmission, the speed and reliability of algorithms and technical solutions decreases [6, 7].

Therefore, it is necessary to improve the data transfer process, and in this article, it is proposed to use knowledge about the data and messages transmitted, which are considered as a certain structure with its own characteristics, and not just as an encoded binary data stream.

1.2 Scientific Researches and Issues Analysis

In order to better research the achievements in the field of error control coding, the articles and developments concerning the combination of different coding methods, adaptive coding and peculiarities of their implementation, as well as modelling of their operation were considered [8–10]. In particular, article [8] presents the usage of turbo codes in power-line communication channel networks (PLC). An example of adaptive codes usage to correct the errors is given in the article [9], which also emphasizes the effectiveness of adaptive algorithms utilizing depending on the current state of a channel. The article [10] presents the features of the implementation of error control coding based on FPGA boards. It should be noted that such implementations are higher in complexity, but their advantage is the greater speed of information processing through the use of parallel computing, which is provided by FPGA.

In addition, the peculiarities of data processing with the use of protocols of the OSI model are considered, in order to utilize as much information about the transmitted data as possible to increase the probability of successful message transmission over network channels [11]. The article [11] states and substantiates the importance to apply algorithms for search and correction of errors during data transmission not only in large and enterprise networks, but also in personal and small user networks.

It is worth noting that this article considers the encoding of data without the loss of information (i.e. it is not about audio or video streams where the partial data loss is permissible).

1.3 The Goals

The goal is to propose and test a more integrated approach to data preparation before transmission over network channels, which includes analysis of the message structure and data contained in the message. The reliability of the transmission of structured data over network channels should be increased with the help of the revealed regularities. The research outlines the effectiveness of the content analysis of structural peculiarities and messages in order to use the results of analysis to improve the performance factors of data transmission. This article proposes to improve existing methods of data encoding by using of information about the content and nature of the messages themselves, which in theory can increase the reliability of the transmission of structured data.

2 Method of Adaptive Error Control Coding

2.1 The Typical Composition of Structured Data

Abstracting from the formats and implementations (JSON, XML, ProtoBuf, etc.), they are united by one: messages are divided into components, which are often called "tokens". These components have a certain semantic meaning (often even the corresponding token name is specified in the message itself), usually unique within the message type, and a set of valid values for each token (type). An information system can have any number of message types and any number of tokens in one message: their number depends on the complexity of the system. In different programming languages, such messages are mapped in different data structures, but in most cases, they are classes (such as in Java) and structures (such as in C). The messages are serialized before being sent over the network channels, and deserialized after receipt.

The previous article [12] discussed a way to use all these peculiarities of structured messages to improve the quality of data transmission over the network channels. In this article we consider the features and implementation details of the proposed method.

2.2 The Client–server Communication

Let's suppose that the information system is built on the principle of client–server architecture. The server responds to the client requests, and there are $k > 0$ types of requests in the system. Each request has its own request structure, and a corresponding unique response structure. All requests and responses to requests are the structured messages that consist of a set of tokens. The scheme of a request (ask) with the index k has the form of $Ak = \{a_{k1}, a_{k2}, ..., a_{kn}\}$, $n > 0$. The corresponding scheme of reply (response) is: $R_k = \{r_{k1}, r_{k2}, ..., r_{km}\}$, $m > 0$. It should be noted that the number of tokens and their type in the request and response may differ. In addition, in more or less complex systems, the tokens are hierarchical and may themselves contain embodied messages, i.e., for example, the response to request number 1 may look like this: $R_1 = \{r_{11}, r_{12}\}$, where token r_{11} is a numerical identifier and token r_{12} is composite: $r_{12} = S\{s_1, s_2, ...\}$.

2.3 Dynamic Subtypes

Let's use the above features of structured messages and requests in order to increase the reliability of data transmission over a network channel. The concept of a "dynamic subtype" is introduced. This is statistical information about the data transmitted by a specific token. It may change over time and be supplemented. In the simplest case, we consider the numerical type of token (int, 32 bits). If a certain number of messages containing a given numeric token and the token value were in the range (100–150) were transmitted for a certain period of time, it means that the current dynamic subtype of the token is a number encoded by $w = 6$ bits of information (0–50) with an offset equal to $s = 100$. Thus, such a simple dynamic subtype for a numeric data type can be encoded by two numbers: the number of bits and the offset. Let's consider another case, for example, for text strings. If a text token contained only a small set of constants (enum), the number of which does not exceed 256, then the dynamic subtype can encode all text values with a numeric single-byte value. In this case, such a dynamic subtype will be encoded with a dictionary, which will contain "key"–"value" pairs, where the key will be a 1-byte number, and the value—the corresponding text string. Another way to encode a text (or byte) string is to use three values: an array of valid admissible characters A and the length of a byte string $\{L_{\min}, L_{\max}\}$.

The current state of the dynamic subtype for each token can be controlled in three ways:

- for each client of the information system separately;
- for all clients in general;
- for each client separately with a timeout (TTL).

In the first case, during the encoding/decoding process, it is possible to take into account the peculiarities of each client device separately. The disadvantage is the greater complexity of the algorithm and greater requirements for computing resources and memory of the device, especially on the server side, because it must store information about dynamic subtypes for each client, and the clients must be identified by a unique identifier (e.g., UUID).

In the second case, the implementation is simpler, but the operational efficiency of the method will be lower. The third way is a compromise between the first two methods. This paper considers the simplest option, i.e., shared state of dynamic subtypes for all client devices.

2.4 Method of Encoding

So, let's have a message of a certain type, which consists of a finite set of tokens. We introduce the concept of message structure history, which contains the information about the current dynamic subtypes of tokens, as well as information about the old subtypes, because synchronization between hosts does not happen instantly. The history can be represented as a list or an array, where the index of an element is its identifier. The main requirements for such a list are:

- the ability to quickly make changes (add a new history element in case of a subtype change detection with one of the message tokens);
- the possibility of direct and fast access to each history element with constant complexity O(1).

The general linked lists are not suitable for the current task (because they do not meet the second requirement), so it is better to use the lists based on arrays, which are expanded by allocating a new array, if necessary, but with a certain growth rate calculated by the formula: $S_{new} = 3 \cdot S_{old}$.

The initial value is $S_{\text{old}} = 10$. It is also possible to use a hash table, which in most cases enables to add and search for elements with almost constant complexity $O(1)$, provided the correct hash function is selected. The hash function should be calculated quickly and has the form of $y = f(x)$, where x is an array of data of arbitrary size, and y is a number/array of fixed size. The number of collisions in the calculation should be as small as possible. That is, the distribution of the function values should ideally approach a uniform distribution [13].

Also, in order to verify the correctness of the data contained in the message, it is necessary to add a checksum to it, with which it will be possible to answer the question with a high probability: whether the decoding process is completed successfully or not. There are many methods for calculating of checksums. The parity bit is the simplest option, which consists of adding one bit to the message, which is equal to "1", if the number of units is even, and "0"—otherwise. In terms of coding theory, the parity bit can be considered as a code $(n, n-1)$ with matrices:

$$H_{1,n} = \begin{pmatrix} 1 & 1 & \cdots & 1 \end{pmatrix}$$
$$G_{n-1,n} = \begin{pmatrix} 1 & 0 & \cdots 0 & 1 \\ 0 & 1 & \cdots 0 & 1 \\ \cdots & \cdots & \cdots 0 & 1 \\ 0 & 0 & \cdots 1 & 1 \end{pmatrix} \quad (1)$$

It is used, for example, in the UART protocol. But it is not reliable, and can only detect an error in one bit [14].

IP packets use a 16-bit complement code as a checksum. The header is divided into 16-bit blocks; their sum is calculated. If there are transfers to a new bit after the calculation, they are added to the amount. This method is more reliable and is already used in the IP protocol at the network level of the OSI model.

Then the typical structure of the encoded message will look like (Fig. 1).

The error control coding method performed by the $\text{encode}_\text{N}(\ldots)$ function can be any method with arguments:

- data to be encoded (d);
- number of additional bits for encoding (Δc).

Furthermore, if the encoding method does not provide an easy way to set and change the code parameters, another way to increase the probability of successful decoding can be used by engaging the callback function, which will be called by the encoding algorithm in case the impossible decoding of a part of the message is detected. The implementation of such a function in the simplest case involves forecasting based on the current scheme of dynamic subtypes, possible variants of the symbol to be decoded, checking the result with a checksum. The disadvantage of this method is that it is necessary to change the decoding algorithm. Therefore, the usage of the first variant is considered. The general algorithm is depicted in Fig. 2.

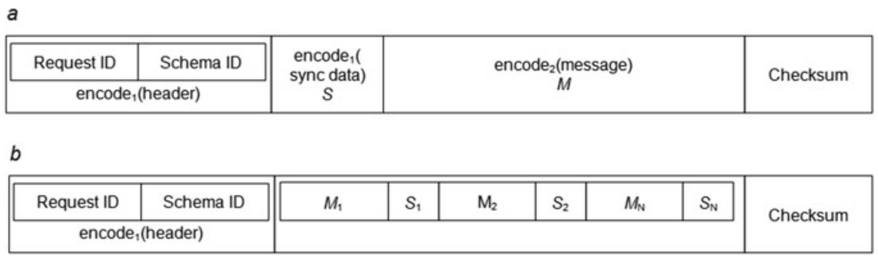

Fig. 1 Logical structure (**a**), physical structure (**b**)

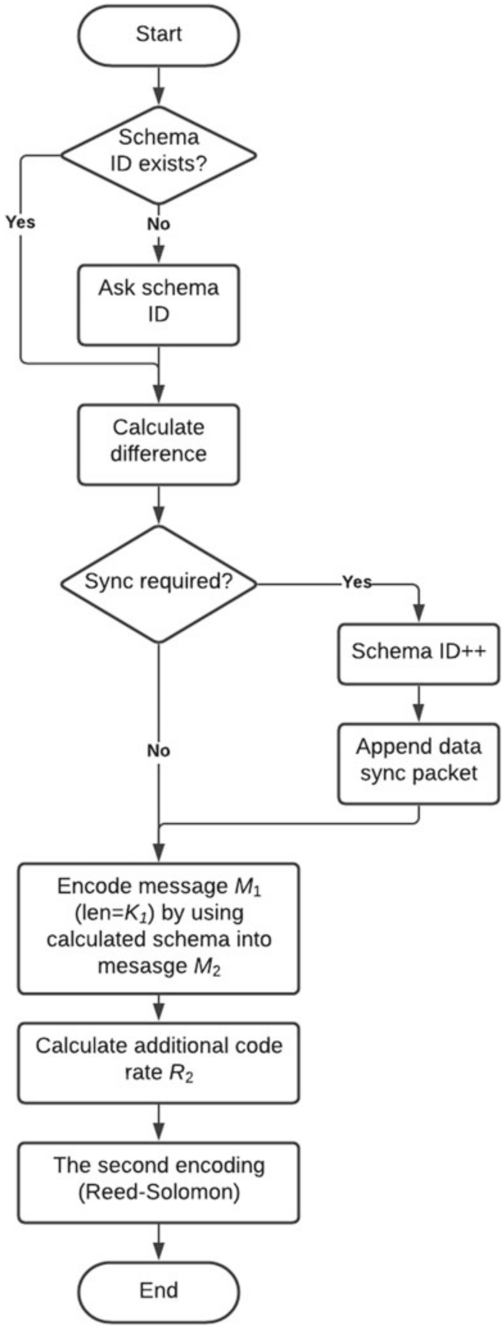
Start
Schema ID exists?
Yes
No
Ask schema ID
Calculate difference
Sync required?
Yes
No
Schema ID++
Append data sync packet
Encode message M_1 (len=K_1) by using calculated schema into mesasge M_2
Calculate additional code rate R_2
The second encoding (Reed-Solomon)
End

Fig. 2 Block scheme of message transmission algorithm

We consider the results based on the Reed-Solomon code (implementation taken from the ZXing Open Source library) and the communication protocol of the ThingSpeak service for Internet of Things devices, developed by IoBridge company (now purchased by MathWorks).

The Reed-Solomon codes are a partial case of BCH codes. The BCH codes are the cyclic codes of some length n over the field GF(q) with a distance d, under the condition that for some value of $b \geq 0$, the generator polynomial is:

$$g(x) = \mathrm{lcm}\{M_i(x),\ i = b, b+1, \ldots, b+d-2\} \tag{2}$$

If we decrease the exponent of the min. polynomials that produce the generating polynomial of the cyclic code, the redundancy of the BCH code also decreases. The exponent cannot be less than one, and it is equal to one if x takes a value in the same field as the field GF(2^m), which is used to build a check matrix of code. In such a field, the minimal polynomial of the element β is $x - \beta$. Since the exponents of the variable x correspond to the position of the codeword, the length of the code must not exceed the number of elements of the multiplicative group of the field.

The Reed-Solomon codes are often used as component codes in cascade constructions. In the method considered in this example, there is no such structure in the usual sense, but there is an add-on over the code in the form of a variable parameter of the code rate and pre-conversion of the message to a more efficient format, which increases the probability of the successful data transmission.

The advantage of the BCH codes and the Reed-Solomon codes in particular is the simplicity of their encoding and decoding algorithm in the channel with hard decision. The disadvantage is the lack of a reasonable decoding algorithm in channels with soft solutions. But this disadvantage is partially eliminated by usage of the method of “minimum generalized distance”, which is reduced to multiple decoding in the channel with hard decision [15, 16].

If the division of the information message into parts and the alternation of encoded tokens with the encoded scheme is not taken into consideration, the procedure can be clearly depicted as follows (Fig. 3):

$$R_1 = K_1/N;\ R_2 = K_2 + K_\mathrm{S}/K_\mathrm{N};\ \Delta C = K_1 - K_2 - K_\mathrm{S}$$

That’s why:

$$R_2 = (K_1 - \Delta C)/N$$

where

K_1	the number of data bits of message encoded by the Reed-Solomon encoder, bits;
N	the total number of bits in the message, bits;

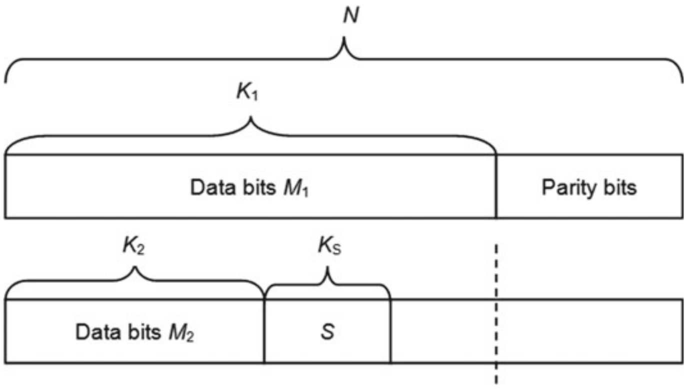

Fig. 3 The procedure of calculating of the amount of possible redundant bits of information

K_2	the number of data bits encoded by state encoder (message without state and schema), bits;
K_S	the number of bits representing state and schema, bits;
ΔC	the amount of space which can be used for adjusted error correction code;
R_1	the code rate of the origin Reed-Solomon code;
R_2	the max possible code rate after adjusting encoder parameters.

Let’s take the following Reed-Solomon code: (15,13), GF(2^4) with the corresponding generator polynomials:

$$(15, 13) : g(x) = x^2 + 6x + 8 \qquad (3)$$

The minimum polynomial is equal to: $\alpha = 2$. The code (15,13) can correct 1 error ($d = 3$). Since the code in fact operates not with bits, but with their sets of 4, one error is equated to an error in any bit in the set (or to an error in all bits together, i.e., the number of errors in the set can be from 0 to 4, but within one set their number is equal to one error), which improves the corrective capability of the code.

We take a set of the following messages of the same size:

```
1) 22 68 22 3a 31 2c 22 74 22 3a 31 37 35 2c 22 77 22 3a 34 35 7d
2) 22 68 22 3a 32 2c 22 74 22 3a 31 36 33 2c 22 77 22 3a 35 30 7d
3) 22 68 22 3a 32 2c 22 74 22 3a 31 32 32 2c 22 77 22 3a 34 34 7d
4) 22 68 22 3a 33 2c 22 74 22 3a 31 37 38 2c 22 77 22 3a 33 38 7d
5) 22 68 22 3a 33 2c 22 74 22 3a 32 30 31 2c 22 77 22 3a 33 39 7d
6) 22 68 22 3a 31 2c 22 74 22 3a 32 30 32 2c 22 77 22 3a 36 30 7d
7) 22 68 22 3a 31 2c 22 74 22 3a 31 37 36 2c 22 77 22 3a 34 38 7d
```

If the extra characters are removed, we obtain the token IDs and values:

```
1) 68 31 74 31 37 35 77 34 35 7d
2) 68 32 74 31 36 33 77 35 30 7d
3) 68 32 74 31 32 32 77 34 34 7d
4) 68 33 74 31 37 38 77 33 38 7d
5) 68 33 74 32 30 31 77 33 39 7d
6) 68 31 74 32 30 32 77 36 30 7d
7) 68 31 74 31 37 36 77 34 38 7d
```

The length of seven messages is 560 bits.

Let's construct the scheme of dynamic types of tokens for a set of messages (let the identifier be equal to 1). We have tokens with keys 0×68, 0×74 and 0×77, which can be matched to single-byte identifiers of the varint type (0×1, 0×2, 0×3). The token subtype 0×1 has a value in the range [0; 2] (0×2) with an offset of 0×31.The token subtype 0×74 has a value in the range [122; 202] (if we convert from ASCII to a decimal number) or if we specify an offset—then it is [0; 80] (0×60) with an offset of 122 ($0 \times 7A$). Performing a similar operation for the third token, we obtain the range [0; 22] (0×16) with an offset of 38 (0×26). So, the message scheme will look like:

Token ID	Origin Name	Type	Max	Offset
0x1	0x68	0x1	0x2	0x31
0x2	0x74	0x1	0x50	0x7A
0x3	0x77	0x1	0x16	0x26

If all numeric values are encoded with varint, and the text values are encoded with the text strings ending in zero-byte 0×0, and to indicate the number of tokens at the beginning of the scheme, the resulting scheme will look as follows:

```
01 | 03 | 68 00 01 02 31 | 74 00 01 50 7A | 77 00 01 16 26
```

The scheme is transmitted once, and the messages encoded using this scheme will look like this (the scheme number comes first, then it is the message):

```
01 00 35 07 01 29 0C 01 00 06 02 38 00 02 4F 01 00 50 16 00 36 0A
```

So we have: an original message size—560 bit. The size of the scheme—136 bit. The size of the resulting message is 168 bits without an identifier.

The first data transfer will contain a synchronization packet, so its size will be a maximum of $176 + 136 = 312$ bits of information. If the scheme does not change, then only the identifier is transmitted instead of the scheme, and therefore it is necessary to transfer $168 + 8 = 176$ bits of information. Hence, we have the values of the parameters: $K_1 = 560$, $K_2 = 176$, $K_S = 136$, $K_2 + K_S = 312$.

We encode the original message with the Reed-Solomon code, GF(24), $\alpha = 2$ (formula 4). Dividing the data array into blocks of 13 information bits, we obtain the following data symbols:

```
 6   8   3   1   7   4   3   1   3   7   3   5   7
 7   3   4   3   5   7  13   6   8   3   2   7   4
 3   1   3   6   3   3   7   7   3   5   3   0   7
13   6   8   3   2   7   4   3   1   3   2   3   2
 7   7   3   4   3   4   7  13   6   8   3   3   7
 4   3   1   3   7   3   8   7   7   3   3   3   8
 7  13   6   8   3   3   7   4   3   2   3   0   3
 1   7   7   3   3   3   9   7  13   6   8   3   1
 7   4   3   2   3   0   3   2   7   7   3   6   3
 0   7  13   6   8   3   1   7   4   3   1   3   7
 3   6   7   7   3   4   3   8   7  13   0   0   0
```

The result of encoding with the above code will look like:

```
 6   8   3   1   7   4   3   1   3   7   3   5   7   6  15
 7   3   4   3   5   7  13   6   8   3   2   7   4  15  11
 3   1   3   6   3   3   7   7   3   5   3   0   7   2   9
13   6   8   3   2   7   4   3   1   3   2   3   2   7  14
 7   7   3   4   3   4   7  13   6   8   3   3   7  13  10
 4   3   1   3   7   3   8   7   7   3   3   3   8  11  12
 7  13   6   8   3   3   7   4   3   2   3   0   3   6   1
 1   7   7   3   3   3   9   7  13   6   8   3   1   6   1
 7   4   3   2   3   0   3   2   7   7   3   6   3  12   1
 0   7  13   6   8   3   1   7   4   3   1   3   7   2  13
 3   6   7   7   3   4   3   8   7  13   0   0   0   9   3
```

The resulting size of the encoded message obviously increases by the number of redundant symbols, and for this code—by 2 symbols for each block. The resulting encoded message size is 165 symbols, or $165 \cdot 4 = 660$ bits. Where 80 bits are parity symbols. If we take the message with the scheme of subtypes, it contains 312 data bits. Therefore, in an equivalent size encoded message can include $660 - 312 = 348$ bits that can be used to improve the transmission reliability.

Code word length $N = 15$. $N_{total} = 660$—total number of bits of encoded message, $k_{total} = 312$—the total number of data bits, $V = N_{total}/N = 660/15 = 44$—the total number of code words (each code word = 15 bits).

So each codeword produced by the adjusted encoder should have minimum of 8 data bits:

$k = k_{total}/V = 312/44 = 8$ (upward rounding)—the minimal amount of data bits in each codeword.

So, we can use max. possible code (15,8) but such code does not satisfy the condition of $(n - k \bmod 2) = 0$. Therefore, it is necessary to round up, so the corresponding code that can be used is (15,9).

$$g(x) = x^6 + 7x^5 + 9x^4 + 3x^3 + 12x^2 + 10x + 12 \tag{4}$$

Table 1 Results for the example above

	$encode_1$[(15,13), message]	$encode_1$[(15,9), $encode_2$(message)]
Length, bits	660	540
Max acceptable percentage of errors, %	6.7	20

This code can correct 3 errors (d = 7) instead of 1. So, its corrective capability is three times greater. This means that the probability of successful transmission of a data packet encoded with 1) the use of information about the structure of the message content and 2) the Reed Solomon's code (15,9) is much higher than a data packet of the same size but encoded only by code (15,13):

```
 0    1    0    3    6    8    0    0    0    1    0    2    3
 1    7    4    0    0    0    1    5    0    7   10    7    7
 0    0    0    1    1    6    2    6    0    1    0    0    3
 5    0    7    0    1    2    9    0   12    0    1    0    0
 0    6    0    2    3    8    0    0    0    2    4   15    0
 1    0    0    5    0    1    6    0    0    3    6    0   10
 0    0    0
```

The result of coding (Table 1).

```
 0     1     0     3     6     8     0     0     0    14     9     9     5
12     0     1     0     2     3     1     7     4     0     0     9     4
10     1     3     4     0     1     5     0     7    10     7     7     0
 6    15     3     6     1    14     0     0     1     1     6     2     6
 0     1     3     3    15    11    15    15     0     0     3     5     0
 7     0     1     2     8     8    15     0     9    12     9     0    12
 0     1     0     0     0     6    10    12     8     0    14     1     0
 2     3     8     0     0     0     2     4    10    12     0     1     2
12    15     0     1     0     0     5     0     1     6    11     2     1
 0     5    11     0     0     3     6     0    10     0     0     0     0
 0     4     4    10    10
```

The function of code parameters selecting in this case is obviously a step function. Therefore, the value of additional corrective capability (as a percentage) depends on the successful result of dividing the message into parts. In particular, if the original message was N bytes in size, and the internal encoder/decoder supports codes (15,x) in this case, the function is depicted in Fig. 4.

Let's explain the Fig. 4 depicted above. It represents an additional portion of errors in the message which can be safely detected and corrected after receiving the message by decoder, depending of amount of data which has been encoded by the

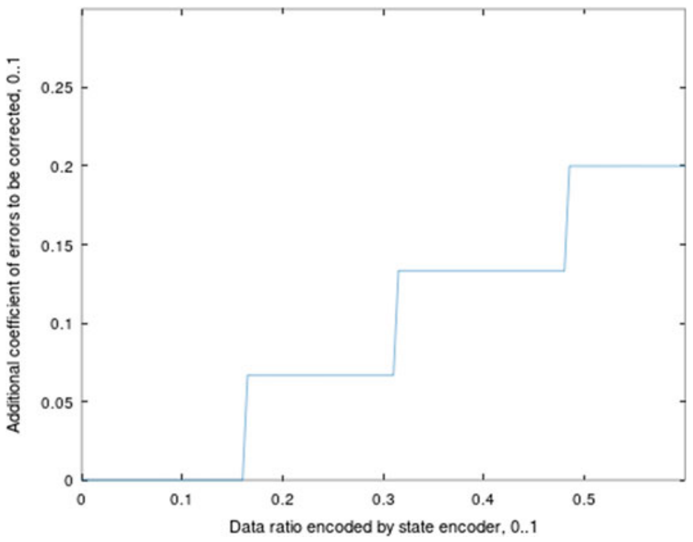

Fig. 4 Additional acceptable percentage of errors which can be corrected

encoder$_2$ (state encoder). For example, let's have an origin message containing 1001 bits. Reed-Solomon (15,13)-encoder produces encoded message with the length of 1155 bits which can be transmitted over network channel. State encoder can extend the amount of space within these 1155 bits for additional parity bits by reorganizing the message content. If state encoder reports to the Reed-Solomon encoder that there are additional $1001 \cdot 0.16 = 160$ bits which can be used for parity bits then the Reed-Solomon encoder may generate code with decreased code rate, e.g. (15,11) instead of (15,13), but without increasing the amount of data to be transmitted. So within the same 1155 bits the acceptable amount of errors which can be safely corrected by the decoder are increased by 6.6%. Please note that the Fig. 4 shows the theoretical limit, actually there are necessarily additional synchronization packets that increase the number of bits of information required for decoding during the transmission.

3 Conclusions and Suggestions

The proposed method of adaptive error control coding analyzes and prepares data for transmission over communication channels by combining 2 encoders: Reed-Solomon encoder (actually it can be replaced by any parameterized encoder) and state encoder. The reliability of the transmission of structured data over network channel is increased in case of using this combined approach as the corresponding decoder may accept and correct more errors than in case of using only Reed-Solomon encoder.

1. The article presents the possibility of effective use of the considered coding method by example: (a) the Reed-Solomon code and (b) encoding the message structure. The possibility to increase the corrective capability provided by the proposed encoding method was tested using a test dataset and the package "communications" in GNU Octave software [17].
2. This method can be combined with various encoding methods, in particular, any other code with the parameters which can be parameterized is possible to use instead of the Reed-Solomon code. In the simplest case, this can be an ordinary linear code (if the goal is to use as simple encoding/decoding algorithm as possible, which can be present in devices with limited resources) or other more complex codes, such as from the family of convolutional codes or turbo codes.
3. The further research can be carried out with the study of the coding efficiency for the data of different types and structures, as the choice of parameters depends on the results of the breakdown of the original messages into parts, which in turn may be different, depending on the internal structure of messages.
4. The algorithm for finding and selecting of tokens can be theoretically improved and optimized for faster operation. It can also be made more versatile by dividing it into two parts by analogy with the approach adopted in the Apache Thrift libraries: (a) token encoding/decoding; (b) classification of subtypes and generation of the range of acceptable values.
5. In addition, depending on the specific implementation of the method, other data structures (binary trees, arrays, etc.) can be used for implementation in different computing architectures. There is a theoretical possibility of parallel computing usage or even full-fledged parallelization based on FPGA.

References

1. Hemanth G (2020) A study on the Internet of Things. Int J Eng Trends Appl 7(4):5–9
2. Ansere JA, Han G, Liu L, Peng Y, Kamal M (2020) Optimal resource allocation in energy-efficient Internet-of-Things networks with imperfect CSI. IEEE Internet of Things J 7(6):5401–5411
3. Alberti AM, Scarpioni GD, Magalhaes VJ, Cerqueira A (2019) Advancing NovaGenesis architecture towards future Internet of Things. IEEE Internet of Things J 6(1):215–229
4. Koivo H, Elmusrati M (2009) Systems engineering in wireless communications. John Wiley & Sons, Chichester, U.K, pp 242–243
5. Plummer DC. RFC826: an ethernet address resolution protocol. In: Internet standard. https://datatracker.ietf.org/doc/html/rfc826
6. Lakshman TV, Madhow U, Suter B (2000) TCP/IP performance with random loss and bidirectional congestion. IEEE/ACM Trans Netw 8(5):541–555
7. Iliev TB, Hristov GV (2007) Simulation estimation of the forward error correction of turbo codes. In: 2007 8th International conference on telecommunications in modern satellite, cable and broadcasting services, pp 521–524
8. Abd-Alaziz W, Mei Z, Johnston M, Le Goff S (2017) Non-binary turbo-coded OFDM-PLC system in the presence of impulsive noise. In: 25th European signal processing conference (EUSIPCO), pp 2576–2580

9. Sunghwan K (2015) Adaptive FEC codes suitable for variable dimming values in visible light communication. IEEE Photonics Technol Lett 27(9):967–969
10. Zinchenko MY, Levadniy AM, Grebenko YA (2019) Concatenated error correction code implementation on FPGA. In: 2019 Systems of signal synchronization, generating and processing in telecommunications, pp 1–4
11. Pollard JK (2007) Packet error correction in personal area networks. In: First Asia international conference on modelling and simulation, pp 287–291
12. Andrushchenko R, Zaitsev S, Druzhynin O, Shelest M (2020) Method of encoding structured messages by using state vectors. In: Mathematical modeling and simulation of systems, pp 144–153
13. Mailund T (2019) The joys of hashing: hash table programming with C. Apress, Berkeley, California
14. Valvano JW (2014) Embedded systems: introduction to ARM® Cortex(TM)-M microcontrollers. Austin, Texas, pp 377–382
15. Kudriashov BD (2016) Foundations of coding theory, pp 70–187
16. Wesołowski K (2002) Mobile communication systems. J. Wiley, New York, pp 30–35
17. GNU Octave official web-site. https://www.gnu.org/software/octave/index