Application of an Adaptive Inversion Frequencies Algorithm for Router Bandwidth Improvement

[1]Evgeniy Kravtsunov, [1]Timur Mustafin,
[2]Andrei Tchernykh, [1]Valery Perekatov, [1]Alexander Drozdov
[1]MCST, Moscow, Russia
{kravtsunov_e, Timur.R.Mustafin, perekatov}@mcst.ru
alexander.y.drozdov@gmail.com
[2]CICESE Research Center, Ensenada, Baja California, México. chernykh@cicese.mx

Abstract

In this article, we study the practical application of the inversion frequencies algorithm for the compression of IP network data. We present a modified version of the algorithm with adaptability. We show that our algorithm can be effectively used for DNS/UDP traffic compression, allowing a 3-15% increase in router bandwidth.

Keywords. *bandwidth, Compression. Inversion Frequencies Algorithm*

1 Introduction

The Inversion Frequencies Algorithm (IF) is used to obtain an output sequence, which is better compressible than the input sequence. The IF scans the input sequence for each alphabet symbol, and produces zero output values for the last symbol of the alphabet. Since a sequence of zeroes does not give any information, it can be omitted. Therefore, the output sequence of the IF stage is shorter than the input sequence.

On the other hand, the IF adds more overhead for the transmission of the symbol distribution or terminator symbols, which makes IF less attractive for small amounts of compressible data.

In this paper, we propose and analyze a new algorithm to increase router bandwidth and data transmission speed using compression of IP network data. The aim was to implement lossless data compression meeting the following requirements:

- The compression algorithm should encode the stream in one pass.
- The amount of main memory required to implement the algorithm, should not exceed 32 KB.
- The costs associated with the transfer of the control information needed to decode the data should be minimized.
- The encoding time for one data block must not exceed the network transmission time saved by the compression.

An additional purpose of the study is to determine the type of network traffic, for which the use of this compression algorithm is effective. The compression algorithm must complete data stream coding in one pass of the input data, which corresponds to an algorithm complexity of *O(n).* The Stream compression and memory minimization requirements mean that the input stream cannot be modeled as a source with memory [1]. This means that compression algorithms based on search coincident substrings and dynamic dictionary [2] cannot be used as solution of the problem. The input stream has to be modeled as a source without memory.

The use of the general interval transformation algorithm is theoretically justified for this kind of sources. For sources without memory, [3] shows that general interval transformation encoding with Rice-Golomb codes provides good compression ratio, which tends to approximate the theoretical limit of compression. Practical implementation of the general interval transformation called inverted frequency coding (IF-transformation) is described in [4].

The implementation in [4] is promising; based on the standard test suite compression results, and an analysis of the proposed algorithm. However, it cannot be used as solution for our problem. To provide an efficient compression for small blocks of the input data, we propose the following improvements: adaptability of IF- transformation, and supporting alphabet letter sizes of 2 and 4 bits.

The adaptability allows an improvement of compression ratio, making it almost equal to the theoretical limit. Supporting letters of sizes 2 and 4 bits minimizes the costs associated with the additional information needed for decoding and transmitting the compressed code.

Compression efficiency for the algorithm is studied based on the Calgary Corpus standard test, and on the real data obtained on the collecting of outgoing internet traffic of MCST. Measurements made on the Calgary Corpus test confirmed the improvement of the compression ratio by adding adaptability to the algorithm.

Measurements on real data have revealed traffic types that can use our small blocks data compression algorithm. Based on these positive results, we have developed a generalized description of the main functions of the algorithm, which can be used to develop an instruction set and internal logic for a compressing ASIC (Application-specific integrated circuit.)

2 Compression Algorithm

The compression algorithm is based on the inverted frequency transformation by Rice-Golomb encoding. Detailed description of the inverted frequency transformation algorithm is given in [Kravtsunov]. Let us present the main improvements related to the adaptability of inverted frequency transformation. The main features of the algorithm are the following:

Alphabet A[N] is a set of symbols (letters) of the text. Each alphabet symbol can be represented by 8, 4 or 2 bits. The Power of alphabet N is a quantity of letters (symbols). So, if each symbol is represented by 8 bits, the power of alphabet is $N=2^8=256$ symbols. For 2-bit and 4-bit symbols power of the alphabet is 4 and 16 symbols respectively.

Entropy Q is the minimal quantity of bits necessary for encoding one symbol of the text, with certain statistics of the symbols. Entropy defines the theoretical limit of the compression. This limit of compression, K, is calculated with the next equation:

$$K = \left(1 - \frac{Q}{S(N)}\right) * 100\% \qquad (1)$$

Where N is the power of alphabet, *S(N)* is the number of bits for representation of one symbol of the alphabet with power equals to N.

Alphabetical order is the order of symbols in which $F(A[N])>F(A[n+1])$, where F(A[n]) is a frequency of symbol A[N] appearing in the text, $F(A[N+1])$ is a frequency of symbol $A[N+1]$ appearing in the text, n belongs to interval $[0;N-1)$.

Lexicographically older symbol, the symbol $A[i]$ is lexicographically older than the $A[j]$ symbol of the $A[N]$ alphabet, if $i > j$ and the $A[N]$ symbols are sorted in alphabetical order.

IF transforms a set of text symbols to N sets of offsets values. Offset is a quantity of symbols lexicographically older than the current symbol, which appeared from previous position of current symbol or from the beginning, if current symbol meets at the first time. If the text presents alphabetical order, long series of small offsets values appear, which can be effectively compressed by Rice-Golomb codes. The key point is the alphabetical order of the statistics of the text. In [4], alphabetical order is calculated for each text block of 65536 bytes, and transmitted with the encoded text block.

From the complexity point of view, this method is acceptable. The downside is that with alphabetical order, statistical heterogeneity within the text block is not considered, reducing the compression ratio. Reducing the size of the block in the implementation of [4] does not solve the problem, and increases the costs associated, since we need to transmit the alphabetical order for each block.

This problem is solved by dynamically changing the alphabetical order in the conversion process, in accordance to the changing statistics of the text.

2.1 Dynamic Alphabetical Order Changing

In our algorithm, as in [4], a full binary tree is used for inverted frequency transformation (Figure 1). There is a counter C in each tree node, initialized to zero. The tree leaves are initialized by an A[N] symbol, following an alphabetical order. The G’ value (the number of lexicographically older symbols from the beginning of the text) is set initially to 0.

Direct transformation of each letter of the input text in [4] is performed by the next sequence of actions (Figure 2):

1. Determinate tree leafs, which correspond to input letters. Calculated offset from the beginning of the text, G, is initialized to zero.
2. Visit each node from the selected leaf to the root with 2 actions on each node:
   • Value G is increased by right brother tree node counter value
   • Current tree node counter is increased by 1
3. Offset value is calculated by Offset = G – G’;
4. Previous G’ value for the symbol is replaced with current G value.
5. Offset is transmitted to the Rice-Golomb algorithm input

Our main improvement is to add an alphabet sorting, which changes alphabetic order after encoding each symbol (Figure 3). The Sorting, in decreasing order of appearance counters, is done after the calculated value Offset is passed to the Rice-Golomb algorithm input. The Quicksort algorithm is used for rebuilding. The proposed method is called “Adaptive rebuilding of binary coding tree". The inverse transformation is also performed using the structure of a balanced binary tree. Rebuilding the tree during the reverse transformation is performed after decoding each letter. Sorting criteria in the reverse and forward transformations coincide, which allows you to decode text.

2.2 2 and 4 bits Letter Sizes Support

In [4], the authors considered alphabet power equals to 256. The letters of that alphabet are 8 bit symbols and the offset was represented by an unsigned int type, which size is 4 bytes. An array of offsets numbers comes out from direct transformation too.

In order to obtain the source text from a compressed format, the array of offsets obtained by direct conversion has to be provided as the input of the inverse algorithm.

The array of offsets numbers is sent with bit-code as a sequence of 1024 bytes (256 symbols * sizeof(int)). Thereby, using eight-bit symbols for each text block introduces costs estimated at 1024 bytes. It forbids effective use of the algorithm from [4] for small data block compression.

For solving the problem of small data block compression, we introduce the support of alphabet powers of 4 and 16 symbols, two and four-bit symbols. We also changed the data type for storing offsets numbers from unsigned *int* to unsigned short. The data type has limited maximal block size of input text of 65536 bytes, but for alphabet power reduction it allows decreasing the costs. The costs are equal to 8 and 32 bytes per block.

2.3 Entropy Calculation Function

Entropy is the minimal bit quantity, required for encoding of one input symbol of a text. We implemented an entropy calculation function, which is used for input blocks of any size and alphabet powers of 4, 16 and 256 symbols. It is applied for estimating the compression efficiency. It is also helpful to compare compression ratios against the theoretical limit.

There are two ways to do entropy calculation:

1. Modeling text as a source with memory.
2. Modeling text as a source without memory.

Which one we choose depends on type of compression algorithm. A source with a memory model is focused on algorithms that do not have a fixed alphabet, but instead have a dynamically created dictionary. A source without a memory model is suitable for algorithms which have a constant alphabet. It is not extended in the transformation process by a dictionary.

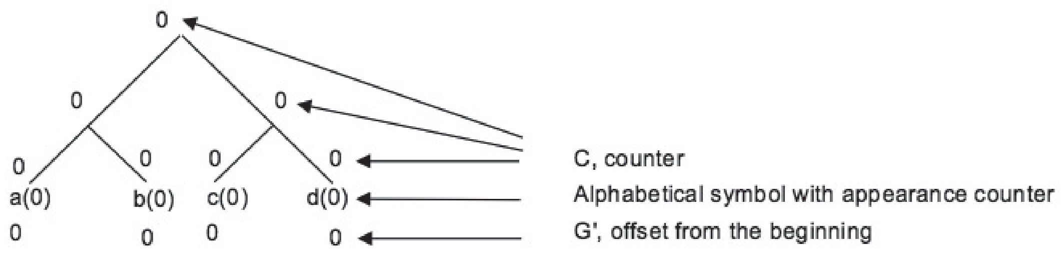
0
0
0
0
0
0
0
a(0)
b(0)
c(0)
d(0)
0
0
0
0
C, counter
Alphabetical symbol with appearance counter
G', offset from the beginning

Figure 1: Full binary tree after initialization

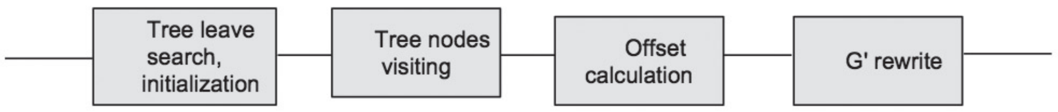

Figure 2: Direct inverted frequency coding

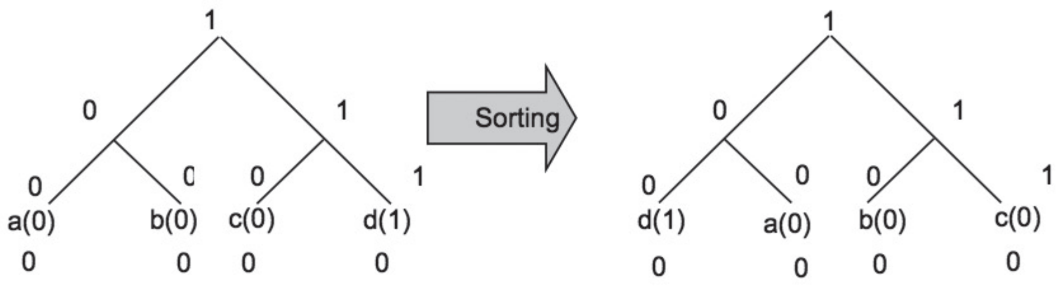

Figure 3: Adaptive rebuilding of binary tree

Because the inverted frequency transformation is based on using of constant alphabet (alphabetic order can change, but the set of symbols is fixed), a model of source without memory is used for estimating the efficiency of the algorithm. In that case, the entropy function H for text with length D looks like:

$$H = \sum_{i=0}^{N} \left(\frac{q_i}{D}\right) * \log_2 \left(\frac{q_i}{D}\right) \qquad (2)$$

Where q_i is the quantity of i letter in text.

3 Measurement Results

In this section, we present the compression ratios of the adaptive inverted frequency transformation with Rice-Golomb coding for two types of test data:

1. Standard test set for compression algorithms Calgary Corpus;
2. Real data – net traffic files of different types.

The measurements on Calgary Corpus aimed to compare of obtained compression ratios with theoretically calculated compression limits. The entropy calculation function was used for theoretical compression limit calculation. Calgary Corpus measurements were also used to measure the effectiveness of adaptive rebuilding of binary coding tree method, and estimated usefulness of alphabet powers of 4 and 16 symbols.

The analysis on real data shows that it is possible to determine traffic type, which can be compressed. The data stream contains a lot of small data blocks with different length and statistics.

3.1 Calgary Corpus Measurements

Table 1 contains the results produced on Calgary Corpus test set by adaptive interval frequency transformation with Rice-Golomb coding. Table 1 shows compression ratios obtained by algorithms with rebuilding and without rebuilding with the theoretical compression limit for each file from the Calgary Corpus test set. The results were obtained with an alphabet power of 256 (symbol size is 8 bits).

Table 1: Comparison of compression ratios in percent of Calgary Corpus files by algorithms without rebuilding and with adaptive rebuilding the alphabet order. Symbol size is 8 bits.

File	Without rebuilding	With rebuilding	Theoretical limit
bib	33,51	33,52	34,99
book1	41,73	42,43	43,41
book2	37,00	39,02	40,09
geo	28,14	28,38	29,42
news	33,65	33,74	35,13
obj1	18,29	22,13	25,65
obj2	19,64	19,83	21,75
paper1	35,14	36,31	37,71
paper2	39,73	40,94	42,48
paper3	36,70	39,95	41,69
paper4	21,26	36,80	41,25
paper5	31,27	33,09	38,30
paper6	33,12	35,36	37,38
pic	77,93	78,69	84,87
progc	32,63	32,52	35,01
progl	38,73	38,48	40,37
progp	37,56	37,18	39,14
trans	29,35	28,58	30,84
Average	34,74	36,50	38,86

Table 1 shows that an adaptive rebuilding of the binary coding tree improves compression of standard files by 2 % and brings compression ratios closer to theoretical limits.

Results with compression ratios for alphabet powers 4 and 16 (symbol sizes equals 2 and 4 bits) are given in Table 2. Results for alphabet power 256 (symbol size is 8 bits) is also given to facilitate comparison.

In Table 2, we find that the theoretical limit of compression value and real compression are reduced as the alphabet power decreases. This situation is not good for compressing big data blocks, where the costs for transmission of control information in compressed format can be neglected. The costs cannot be neglected for smaller data blocks with sizes from 12 to 1500 bytes. Costs obtained with alphabet power 256 do not allow compressing small data blocks. Table 2 shows that a 16 symbols power alphabet provides near to theoretical compression. Hence, we conclude suitability of 16 symbols power alphabet for network traffic compression.

3.2.1 Different Net Traffic Types, DNS Traffic

Table 3 contains results of theoretical compression limits for files with different traffic types. The entropy is calculated for the set obtained by merging all data segments into one. This allowed us to determine compression feasibility of either traffic type.

Type of traffic and compressible DNS traffic were recorded in the file of size 2607767 bytes, consisting of 19102 small packets of different lengths, from 12 to 1500 bytes. This type of traffic represents a suitable set to test the conversion algorithm and Rice-Golomb encoding for small blocks.

For the experiment, data segments of the size specified in the header were extracted from traffic packets. Measurements were carried out on an alphabet capacity of 4 letters (costs are minimal) using the adaptive refinement of the binary tree. For each block of data, the entropy is also calculated. The experimental results are shown in Table 4.

It is clear that despite the theoretical calculations showing that the packets can be compressed (column “entropy”),only 30% of the available sample packs were compressed. This discrepancy is due to the presence of costs, that is, the presence of additional information in the compressed code needed to decode the data. Thus, the total effective compression of network traffic can be achieved by compression of the packets with a good compression ratio, and the remaining packets will be transmitted uncompressed. To determine which packets are suitable for compression, we propose a decision taking method based on a comparison of the calculated values of the entropy of the data segments with the criterion R obtained by the experimental method.

R value defines the boundary values of entropy. In fact, the R value characterizes non-optimality of compression and allows to evaluate the possibility of real compression of the data.

3.2.2 Method of calculating the decision criterion for package compression

The proposed method of deciding about compression is based on the assumption of dependency between the entropy and compression rate. If the entropy is greater than R, it is decided not to compress the package, otherwise it is compressed.

The size of compressed files is calculated in advance. This allows to known what packages have to be compressed. This information allows us to judge the correctness of the decision.

We tested the algorithm on 19102 packets with different values of R ranging from 0.6 to 2. The result of the simulation is presented as a function of R. Figure 4 shows the total size of packets of the network after the application of the algorithm.

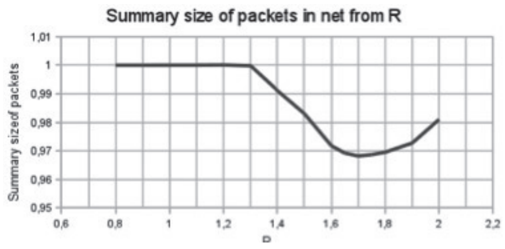

Figure 4: Compressed Size of packets

We show that the optimal value of R is 1.7. This value provides maximum compression.

3.2.3 Compression DNS packet flow using the decision method

Table 5 shows the characteristics of the decision-making method for the DNS compression packages. Total compression ratio of all flow coincides with what is shown in Table 3 for DNS traffic. The discrepancy between the total compression ratio as a set of small packets and entropy presented in Table 3 is due to the fact that the compression algorithm does not have enough statistics of such small packages.

Table 2: Compression ratios of Calgary Corpus files for alphabets with different powers

	Alphabet Power			Theoretical Compression Limits		
File	4 letters	16 letters	256 letters	4 letters	16 letters	256 letters
bib	-2,46	9,67	33,52	2,74	11,77	34,99
book1	-2,53	15,51	42,43	2,45	16,41	43,41
book2	-2,57	14,21	39,02	2,20	15,25	40,09
geo	12,19	14,38	28,38	16,81	17,92	29,42
news	-2,94	11,28	33,74	1,91	12,75	35,13
obj1	6,08	15,40	22,13	9,85	16,25	25,65
obj2	0,13	7,88	19,83	5,02	10,18	21,75
paper1	-2,58	13,77	36,31	2,19	14,70	37,71
paper2	-2,40	15,23	40,94	2,43	16,45	42,48
paper3	-3,11	15,06	39,95	2,41	15,96	41,69
paper4	-2,69	13,96	36,80	2,35	15,26	41,25
paper5	-2,98	12,39	33,09	2,13	14,34	38,30
paper6	-2,67	12,68	35,36	2,10	13,75	37,38
pic	44,20	67,06	78,69	73,82	80,78	84,87
progc	-3,45	10,07	32,52	2,13	12,41	35,01
progl	-3,82	10,21	38,48	1,98	12,11	40,37
progp	-2,25	10,70	37,18	2,87	12,44	39,14
trans	-3,53	6,13	28,58	1,90	8,29	30,84
Average	1,26	15,31	36,50	7,63	17,61	38,86

Table 3: Results of the analysis of various types of network traffic for compressibility

Traffic type	TCP/UDP	Port	Theoretical compression limit: 4 alphabet letters	Theoretical compression limit: 16 alphabet letters
HTTP	TCP	80	0,00	0,50
HTTPS	TCP	443	0,00	0,00
SMTP	TCP	25	0,00	0,00
DNS	UDP	53	14,75	18,75

Table 4: Results of experiments on the compression of data segments packet DNS traffic.

Data segment size	Total	Compressed	Uncompressed	Entropy
< 300 byte	17041	5016	12025	0,83-1,96
300-500 byte	814	489	325	1,29-1,95
> 500	1247	246	1001	1,26-1,97
	19102	5751	13351	0,83-1,97

Table 5: Degree of compression using different decision-making principles

All packets	Method of making a decision with R = 1,7	Using a posteriori knowledge about compression package
1,9	3,18	4,73

4. Hardware Support for Compression

Modern trends in the network to accelerate the traffic to increase the MTU to 9000 and 16000 bytes (Jumbo and SuperJumbo frames) [5], allow to assume that the hardware implementation of adaptive algorithm Literal interval transformation and coding Rice-Golomb compression can be useful for blocks of medium size. For these sizes, one can use the alphabet of power 16, allowing us to compress DNS traffic up to 15%. Hardware implementation of the algorithm should provide the following APIs:

- Initialization of the tree
- Tree traversal - increase counters
- Rebuilding of the tree
- Calculation of bit code conversion Rice –

Golomb
- Adaptation of the Rice – Golomb factor
- The decision based on the value of the entropy

5. Conclusions and Future Work

This article describes the implementation of the algorithm for adaptive Literal interval transformations, which allows compressing small blocks of data.

The implementation uses an adaptive reconstruction of binary tree literally equal interval transformation and alphabets capacity of 4 and 16 characters.

The authors experimentally determined the type of traffic being compressed by the algorithm. A method for estimating the compressibility of the block, based on the theoretical calculation and comparing it with the criteria characterizing the algorithm was presented. APIs for hardware compression are also proposed.

References

1. Krichevsky, R. (1989). Compression and information retrieval. *Radio and Communication*, Moscow.

2. Nelson, M. & Gailly, J. (1996). The Data Compression Book. *M & T Books*, New York.

3. Brailovski, I.V. (2003). Effective Data Compression Using the Generalized Interval Transformations. Ph.D. Thesis, Moscow.

4. Kravtsunov, E.M. (2004). Effective Methods of Data Compression for Embedded Systems. High *Performance Computing and Microprocessors* , ,IMVS RAS, Issue 6 , p. 103 , Moscow.

5. Leitao, B. (2009). Tuning 10Gb network cards in Linux, a basic introduction to concepts used to tune fast network cards. *Proceedings of the Linux Symposium*, p.169, Quebec, Canada.