Joint LZW and Lossless Dictionary-based Bit-Packing Compression Techniques for Congested Network

Lee Chin Kho, Yasuo Tan and Yuto Lim
School of Information Science
Japan Advanced Institute of Science and Technology (JAIST)
1-1 Asahidai, Nomi City, Ishikawa 923-1292, JAPAN

***Abstract*—One of the viable solutions for reducing congestion in networks is data compression. Data compression reduces data size and transmission time. This paper proposes a joint data compression technique to eliminate the redundancy of data in a congested network with limited bandwidth and buffer. The proposed compression techniques combine the Lempel Ziv Welch (LZW) and Lossless Dictionary-based bit-packing (LDBP) compression methods. The LDBP is a novel compression technique that requires lesser additional memory compared to LZW. The joint LZW and LDBP compression techniques will be operating in the routing domain and it consists of two main stages. The first stage is the congestion status prediction. If it is predicted that the particular network will be congested, and to be compressed data packets satisfy the compression conditions, the data packets will be forwarded to the second stage. Based on the available buffer and processing time, the compression technique LZW or LDBP will then be implemented in the second stage. The results show that the redundancy of packets can be eliminated. This further reduces the size of data packets.**

***Index Terms*—congested network, LZW compression, lossless, dictionary-based compression, redundancy**

1. INTRODUCTION

Network congestion is one of the unending problems that depend on the insufficient capacity of the underlying sub-network for the demanded amount of data which is rapidly increased. This growth of demand will eventually go beyond the Service Provider's ability to efficiently cope with the huge data traffic. As a result, the network will tend to face tremendous and unpredictable network congestion. When network congestion occurred, the quality of service (QoS) and energy efficiency in the network will be degraded.

The three possible methods to resolve the network congestion problem are, increasing the physical output, increasing the memory size, and decreasing the incoming data rate. The format two methods can only be done manually and it involves costly system updates. That leaves only the last method of controlling the incoming data rate when needed, not to deteriorate the network performance when congestion does not occur. TCP congestion control variants such as Tahoe, Reno, Vegas, Westwood and the others implement this method to reduce the network congestion. However, the TCP congestion control variants still cannot resolve the problem perfectly. When the data is propagated back to the sender in the case of serious congestion, packet loss might still occur due to collision among nodes. Besides, simply reducing the transmission rate at the transport layer might raise the difficulty in maintaining the network throughput stability. To overcome these shortcomings, data compression can be one of the viable solutions.

The lossless data compression technique which utilizes LZW and LDBP are implemented in this paper. LZW is simple to implement, and has the promising throughput in hardware implementation [1]. However, LZW needs huge additional memory during dictionary construction, which might not be able to be provided by the router or switching devices during the network congestion. LDBP is introduced to overcome this problem. LDBP needs lesser additional memory, but requires higher processing time. Therefore, the joint LZW and LDBP compression technique is proposed in this paper. By adaptively compressing the data with both techniques that is based on the network environment and availability of router's memory, the networks that are going to be congested will surely have a higher probability to be released.

The objective of this paper is to propose a data compression technique for the congested network with limited bandwidth and buffer. Our contributions are:

- A novel data compression technique, LDBP is proposed. This algorithm needs lesser additional memory compare to the LZW during the encoding and decoding process.
- Joint LZW and LDBP data compression technique to overcome the shortcomings of respective compression techniques.
- Implementation the joint LZW and LDBP data compression in edge router or switching device. This allows more redundancy of data to be eliminated, as stated in [2], the data of network traffic collected in their paper contain around 50% of duplicate strings across the packets.

The arrangement of this paper is organized as follows. Section 2 describes the related works of data compression in congested network. Section 3 devotes the system model, definition and notation. Section 4 discusses the joint LZW and LDBP compression technique and algorithm. The results and analysis of joint LZW and LDBP are shown in Section 5. Lastly, this paper is concluded in Section 6.

2. Related Works

There are few on-going researches that implement data compression techniques to release the congested network. In [3], the real time adaptive packet compression scheme is developed to improve the performance of high latency network with limited bandwidth. The scheme implements the zlib compression library for compression and decompression the blocks of aggregated packet. The simulation results showed that the scheme may improve up to 90% of the packet drop rate in a heavy load satellite network. L. S. Tan et. al claimed that this real time adaptive packet compression scheme is more suitable for congested limited bandwidth network.

The adaptive compression-based congestion control technique (ACT) [4] uses both of the lossy and lossless compression techniques to mitigate the congestion problem in the wireless network. The compression techniques that are used in ACT are adaptive pulse code modulation (ADPCM) and run-length coding (RLC). The discrete wavelet transform (DWT) technique is performed to classify the data into groups with different frequencies to create a priority-based congestion control. The experiment results showed that ACT is capable of increasing the network efficiency and ensuring the fairness among nodes.

The aforementioned researches depict that the data compression techniques can be implemented in the low bandwidth communication network and reduce the transmission packets size. Unlikely, these studies do not show the overhead of the compression such as memory and time needed for their system.

3. Preliminaries

A. System Model

Fig. 1 illustrates the system model of deploying the joint LZW and LDBP data compression in a network. The network is comprised of senders, router or gateway, and receiver. The joint LZW and LDBP compression is embedded in the router domain, particular at edge router or gateway. This is because the congestion collapse generally occurs at the 'choke point', where the total incoming data to an edge router or gateway exceeds the outgoing bandwidth. In the Fig. 1, multiple routers are transmitting their data packets to an incoming edge router or gateway causing it to be congested. When the buffer of edge router or gateway is beyond some threshold, the joint LZW and LDBP compression is activated and performed to reduce the data packets size. When the number of data packets that have the same destination reach a threshold, the data packets will be grouped into a block. The compressibility of the data in a block is pre-determined by randomly selecting samples from the data packet to be compressed. If the compression ratio (CR) of these samples is less than the compressible threshold, the block data is transmitted directly. If not, the data in the block will be sent to joint LZW and LDBP compression for encoding and transmitting. If the outgoing edge of router or gateway detects that the received data packets are compressed, the data packets will be decoded.

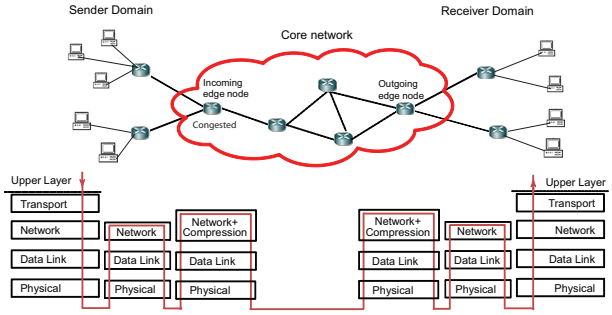

Fig. 1: Architecture of network compression

B. Definitions

Definition 3.1. (Character) *A character (x) is a set of hexadecimal number characters, which uses the characters 0–9 to represent values zero to nine, and A–F (or alternatively a-f) to represent values ten to fifteen. Each character represents four binary bits (or a nibble), which is half of a byte (8 bits).*

Definition 3.2. (Symbol) *A symbol (s) is a group of two or more characters. If the symbol has two characters, then the maximum number of the symbols is 16^2, which is equivalent to 256.*

Definition 3.3. (Symbol length) *A symbol length (l_s) is the number of bits of a symbol. If the symbol has two characters, then the symbol length is 2 characters $\times$ 4 bits = 8 bits.*

Definition 3.4. (Codeword) *A codeword (c) is a group of two or more concatenated symbols. If the codeword has two symbols and each symbol has two characters, then the maximum number of the codewords is 256^2, which is equivalent to 65536.*

Definition 3.5. (Codeword length) *A codeword length (l_c) is the number of bits of a codeword. If the codeword has two symbols and each symbol has two characters, then the codeword length is 2 symbols $\times$ 2 characters $\times$ 4 bits = 16 bits.*

Definition 3.6. (Code) *A code (C) is a mapping from a symbol or a codeword to a set of finite length of binary strings.*

Definition 3.7. (Code length) *A code length (λ) is the length of a code. If the code has λ bits, then we can encode at most 2^λ of symbols and codewords.*

Definition 3.8. (Fixed length code) *A fixed length code is a code such that $\lambda_i = \lambda_j$ for all i,j. For example, symbol, $\{AB\}$ and codewords, $\{9F1B\}$ in fixed length code of 2 bits, the code would be $C(AB) = 00$, and $C(9F1B) = 01$.*

Definition 3.9. (Dictionary) *A dictionary (D) is initialized to contain the single codeword corresponding to all the possible input characters. The dictionary is identical to the input source data.*

Definition 3.10. (Entry) *An entry is a unique codeword that formed from the concatenation of symbols in the dictionary.*

4. JOINT LZW AND LDBP COMPRESSION

The joint LZW and LDBP compression is only activated when the buffer of router or gateway reaches the threshold. In this study, the congestion stages are defined according to buffer level in the router or gateway. The congestion level are is divided into three stages: moderate, serious, and absolute. When the buffer level reaches the threshold of moderate stage, the joint LZW and LDBP compression is activated. At this stage, the available buffer is suitable for LZW to be completed.

In the situation where LZW cannot reduce the buffer level, the congestion will enter serious congestion stage. At this stage, most of the available buffer are being used to store the incoming data packets. The LDBP which needs lesser memory is performed to compress the data packets. However, in the absolute congestion stage, the compression will be halted. This is because the available buffer is not enough for the compression process.

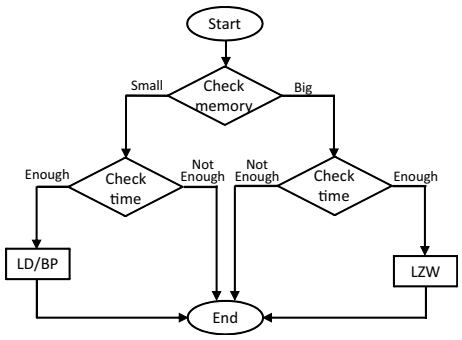

Fig. 2: Process flow of joint LZW and LDBP compression

Fig. 2 shows the process flow of joint LZW and LDBP compression. When the joint LZW and LDBP compression is activated, two parameters are determined, which are the available memory (buffer level) and time (transmission time-out). If the available memory is larger than the previous memory that needs for encoding process, and the time is lesser than threshold time of LZW, the LZW compression will be performed to encode the block of data.

On the other hand, if the available memory is lesser than the previous memory needs for encoding process, and the time needs for the encoding process is longer than time threshold of LDBP, the LDBP compression is performed to encode the block of data.

A. LZW Compression

The LZW compression is a well-known and mature technology that can compress any sort of data (binary or ASCII). The LZW compression is a universal lossless compression. It does not need to know the statistical information of the data being compressed as is necessary with Huffman coding, Fano-Shannon coding, and Arithmetic coding.

The LZW compression compresses the data by using a dictionary to store repetitive codewords that occur in the source data stream. The dictionary is initialized with a set of symbols, i.e the 256 ASCII character as a starting block. This dictionary will be updated when a new phrase is encountered during the compression. For example, in a text document that has many repetition phrases, such as “hello” in the dictionary entry. The “hello”, which is 5 bytes (or 40 bits), can be represented by as few bits (as 9 bits for example). The compression of this codeword is 9/40, which is 23% of the original size of symbols.

The LZW compression can provide a quick long codeword that can be stored in the dictionary, but many of the stored codewords are wasted without being used in the encoding process. When the dictionary is filled with large and frequently used codewords, the dictionary size will be bigger than the source data size. Although the dictionary in the LZW compression can grow without limitation, when the dictionary gets too big, the existing dictionary must be deleted [5]. Then another new block of dictionary is created during the encoding process. Since the encoder and the decoder have the initial dictionary, all new entries in the dictionary are created according to entries in the dictionary that is already exists. The decoder can recreate the dictionary quickly as encoded data is received. In order to decode an entry of a dictionary, the decoder must have received all the prior entries in the block [6].

In summary, the LZW compression has no communication overhead and requires a simple mathematical computational, but the required memory for encoding and decoding is huge. To overcome this problem, the LDBP compression is proposed.

B. LDBP Compression

To reduce the memory requirements in the encoding and decoding process, the dictionary size is limited. In the LDBP compression, the dictionary needs to be transmitted to the decoder for decoding process. The codeword entries of the dictionary is limited to 256. But, this will reduce the compression performance. To overcome this shortcoming, the dictionary is filled with higher repetition codewords first regardless of their position of occurrences. Each of the codewords in the dictionary must fulfil the minimum repetition threshold (R_{thre}). The R_{thre} is given by

$$R_{thre} = \frac{8(l_{cmax}+1)}{(8l_i - \lambda)} \tag{1}$$

The l_{cmax} is the maximum number of codeword length. The l_i is the codeword length that between the maximum and minimum of codeword length, and λ is a code length. The codewords that did not fulfil the repetition threshold condition are removed from the dictionary.

A fixed length code is used in the LDBP. It allows the decoder to access any coded data block in a packet randomly. The codeword boundaries are clear, which is a valuable feature from an engineering viewpoint. For example in [11], [12], the variable length to fixed length codes have been re-evaluated to speed up the search for compressed data.

Similarly, the dictionary in LDBP compression is initialized with a set of symbols. These symbols are corresponding to all possible character. Both encoder and decoder are assumed to

have these symbols in the dictionary with the index from 0 to 255. Therefore, the codeword in the dictionary must start from λ=9 bits. If the λ is set to 9, the dictionary will have 512 entries. Since the first 256 entries is reserved for the symbols, only 256 entries are occupied for the codewords.

The LDBP algorithm consists of three process: dictionary building, encoding and decoding.

1) Dictionary building: The dictionary building algorithm is shown in Algorithm 1. A window that is based on the l_{cmax} is used here. The possible longest codewords are formed from the symbols in the window. The Fig. 3 shows an example of possible longest codeword forming process. In a window of 6 bytes, the first longest possible codeword is formed according to the l_{cmax}, then stored it in the input list and count the number of codeword. The next possible longest codeword is formed according to $l_{cmax} - 1$. If this codeword does exist in the input list, add one into the count of number of codeword. This process is repeated until it reaches the l_{cmin}. Then, the window shifts one byte to the left and starts to form the possible longest codeword again. When the number of codeword reaches the minimum R_{thre}, the codeword is stored into the dictionary. The codewords in the dictionary are sorted in descending order and updated based on the count until the end of data block. The codeword that has the minimum number of codeword will be replaced by the higher number of codeword.

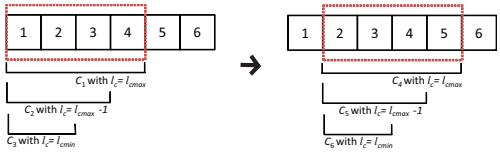

Fig. 3: An example of possible longest codeword forming process

Algorithm 1 Dictionary Building

```
Input window = read (l_cmax − 1)
while (input window.append(read one byte)!=empty) do
    for (codeword=next possible longest codeword) do
        if !(codeword in input list) then
            insert (input list, codeword)
        else
            input list [codeword].count+=l_c
            if (codeword in the dictionary) then
                continue
            end if
            if (input list [codeword].count < R_thre * l_c) then
                continue
            end if
            if (dictionary full and input list [codeword].count==R_thre * l_c) then
                continue
            end if
            if (dictionary full) then
                remove (dictionary,lowest count codeword)
                insert (dictionary, codeword)
            end if
        end if
        remove first byte window
    end for
end while
```

2) Encoding: The encoding algorithm is showed in Algorithm 2. An input window is also used here. If the longest possible codeword in the input window is found in the dictionary, the index of the codeword is output and the bit-packing function will be performed. The bit-packing function packs bits tightly without requiring a lot of adding and shifting. The codeword in the input window is then removed by shifting it to the left. If the longest possible codeword in the input window is not found in the dictionary, the first byte exactly the same is output and the bit-packing function is performed. This time, the input window is shifted one byte to the left to remove the symbol. This process is repeated until the data block is fully encoded.

Algorithm 2 Encoding Process

```
while (input window = read next input window)!=empty do
    for (codeword = next longest possible codeword (input window)) do
        if (codeword exist in dictionary) then
            output (bit packing (found Index))
            remove codeword from input window
        end if
    end for
    output (bit packing(first byte))
    remove first byte from input window
end while
```

3) Decoding: The decoding algorithm is showed in Algorithm 3. The decoding process is exactly revised way of the encoding process. The bit in the dictionary and data are unpacked. If the index of encoded data is higher than 256, output the codeword from the dictionary. If not, output the index.

Algorithm 3 Decoding Process

```
load dictionary
while (true) do
    index = unpack bits (compressed stream)
    if (index = EOstream_flag) then
        exist
    end if
    if (index < 256) then
        output(index)
    else
        output (dictionary [index-256])
    end if
end while
```

5. Numerical Simulation and Discussion

In this paper, the compression ratio (CR) is defined as the ratio between the size of the source data (S_o) and the size of the compressed data S_c, which is expressed as

$$CR = \frac{S_o}{S_c} \tag{2}$$

The processing time and memory for encoding and decoding using the LZW and LDBP is presented in the Table below. The c code of LZW compression in this simulation is developed by V. Antonenko [15].

Test environment: The simulations work is performed using the GNU/Linux 3.13, 32-bit operating system, Intel Core 2 Duo 1.20GHz CPU, 4GB main memory, 160GB 66MHz hard disk. The C code of LZW and LDBP are compiled with gnu C version 4.8.1.

Data Set: Ten test samples are used in the compression test. The sizes and categories of these test samples are given in Table I. These test samples are taken from Canterbury Corpus [14], except the test sample of tulip which is from Archive Compression Test [15].

LDBP Compression Parameter Setting: Code length, $\lambda = 9bits$, Block size =$64KBytes$, Maximum Codeword length, $l_{cmax} = 6$, and Minimum Codeword length, $l_{cmin} = 2$

TABLE I: The Test Sample Details

Test sample	Size (Kbytes)	Category
alice29	152	English text
book2	611	Non-fiction book
Ice10	427	Technical writing
paper1	53	Technical paper
news	377	USENET batch file
obj2	246	Object code
progp	49	Source code
geo	102	Geophysical data
trans	93	Transcript of terminal
tulip	1,153	Photographic picture

TABLE II: The LWZ and LDBP Processing Time (ms)

Test sample	LZW		LDBP	
	Encode	Decode	Encode	Decode
alice29	10.0	2.0	1670.0	41.5
book2	70.0	20.0	6528.5	178.7
Ice10	40.0	10.0	4620.5	114.9
paper1	5.6	1.6	537.9	17.2
news	50.0	10.0	3826.6	129.4
obj2	40.0	10.0	2506.9	18.7
progp	5.0	1.4	541.0	14.6
geo	30.0	7.0	961.6	28.7
trans	10.0	2.0	952.0	28.6
tulip	830.0	210.0	12003.0	970.3

Table II shows the processing time of encoding and decoding for LZW and LDBP compression. Both the processing time is mainly effected by the data format. For example, the encoding time for the test sample of tulip are 83 times higher than alice29, eventhough the data size of tulip is just 7.5 times larger than alice29. In addition, the processing time for LZW is significantly lesser than LDBP. As opposed to LZW, LDBP needs additional time to build the dictionary because it cannot encode while building the dictionary.

TABLE III: The LWZ and LDBP Additional Memory Requirement (Mbytes) and Compression Ratio (CR)

Test sample	Memory (Mbyte)		Compression Ratio	
	LZW	LDBP	LZW	LDBP
alice29	69	2	2.44	1.76
book2	72	5	2.54	1.60
Ice10	71	3	2.62	1.73
paper1	69	3	2.12	1.59
news	71	6	2.13	1.45
obj2	70	6	1.92	1.42
progp	69	2	2.57	1.69
geo	69	7	1.32	1.55
trans	69	2	2.45	1.56
tulip	77	81	0.98	0.94

Table III shows the additional memory requirement and CR for both LZW and LDBP. The additional memory needed by LDBP is at least 10 times lesser than LZW, except test sample of tulip. The LDBP needs an extra 4Mbytes of additional memory compared to LZW in tulip. For the compression performance, the LZW performs better than LDBP in general. The LDBP only performs better in the test sample of geo, which is 0.25 higher than the CR in LZW.

In summary, the LWZ needs lesser processing time and performs better than LDBP. However, the LZW requires larger additional memory compared to LDBP. By applying joint LWZ and LDBP, the tradeoff between these two compressions in releasing the congested can be mitigated.

6. Concluding Remarks

The demand for data transfer might be more than available bandwidth of a sub-network leading to congestion. Therefore, this paper proposed a data compression technique, namely joint LZW and LDBP at the router or gateway (sub-network) to release the congested network. The novel LDBP compression was proposed and described in this paper. The simulation results have proven that the additional memory needed by LDBP compression was significantly lesser than LZW. The LDBP was suitable to be used during the serious congestion stage, while LZW was suitable to be used in the moderate congestion stage. The pre-determination of data compressibility was very important to avoid the uncompressed data, which will worsen the compression performance like the test sample of tulip. Overall, the LZW or LDBP compression can save at least 30% of the network bandwidth if the data flow was text format.

References

[1] T. Welch, “A technique for high-performance data compression,” in *Computer*, vol 17, no. 6, pp. 8–19, 1984.

[2] S. H. Shen, A. Gember, A. Anand, and A. Akella, “Refactoring content overhearing to improve wireless performance,” in 17^{th} *Int. conf. on Mobile computing and networkinh (MobiCom)*, Las Vegas, NV, 2011, pp. 217 – 228.

[3] L. S. Tan, S. P. Lau, and C. E. Tan, “Quality of service enhancement via compression technique for congested low bandwidth network,” in *Int. Conf. on Commun. (MICC)*, Sabah, Malaysia, 2011, pp. 71 – 76.

[4] J. Hyoung and I. B. Jung, “Adaptive-compression based congestion control technique for wireless sensor networks,” in *J. of sensors*, vol. 10, pp. 2920 – 2945, Mar. 2010.

[5] K. C. Barr, and K. Asanovic, “Energy-Aware Lossless Data Compression,” in *ACM Trans. on Comp. Sys.*, vol. 24, no. 3, pp. 250 – 291, Aug. 2006.

[6] C. M. Sadler, and M. Martonosi, “Data compression algorithms for energy-constrained devices in delay tolerant networks,” in *Proc. of Int. Conf. on Embedded networked sensor sys.* New York, USA, pp. 265 – 278, Nov. 2006.

[7] I. H. Witten, R. M. Neal, and J. G. Cleary, “Arithmatic coding for data compression,” in *J. Communication of ACM*, vol. 30, no. 6, pp. 520 – 540, June 1987.

[8] P. Deutsch, “Deflate compressed data format specification”, *RFC 1951*, 1951.

[9] F. Willems, Y. Shtarkov, and T. Tjalkens, “The context-tree weighting method: basic properties,” in *IEEE Trans. Info. Theory*, vol. 41, no. 3, pp. 653 – 664, May 1995.

[10] A. Beirami and F. Fekri, “Results on the redundancy of universal compression for finite-length sequences,” in *IEEE Int. Symp. Info. Theory*, pp. 1504–1508, Aug. 2011.

[11] T. Kida, “Suffix tree based VR coding for compressed pattern matching,” in *Conf. on Data Compression*, Snowbird, UT, pp. 449, Mar. 2009.

[12] S. T. Klein and D. Shapira, “Improved variable to fixed length codes,” in *Lecture Notes in Comp. Sci.*, vol. 5280, pp. 39-50, 2009.

[13] V. Antonenko, ”Implementation of LZW compression algorithm C,” https://code.google.com/p/clzw/source/browse/src/lzw-enc.c?name=69372a470e&r=ace0a939d395d42ac6c16c003fe3c0f5caefda10, access on August 2014

[14] M. Powell, “The canterbury corpus,” http://corpus.canterbury.ac.nz/descriptions/#cantrbry, access on June 2014.

[15] J. Gilchrist, “ Jeff Gilcrist Compression: Archive compression test,” http://compression.ca/act/act-files.html, access on June 2014.