Can a Flight Data Recorder be Placed in a Cloud?

Yair Wiseman
Israel Aerospace Industries
Ben-Gurion International Airport
Lod, Israel
wiseman@cs.biu.ac.il

***Abstract*— Flight Data Recorders (Black Boxes) generate data that is collected on an embedded memory device. A well-known difficulty with these devices is that the embedded memory device runs out of space. The worry of getting into this problematical situation compels the software of the flight data recorder to operate in a watchful mode - it constantly makes efforts to minimize the use of memory space; otherwise a larger flight data recorder will be needed; however a larger flight data recorder can be very problematic because flight data recorders have very rigorous requirements; therefore an enlargement is very expensive and costly. In this paper it is suggested to send the data to a remote cloud, so the flight data recorder memory device will be in actual fact unbounded.**

Keywords*— *IaaS, Flight Data Recorder, Data Compression.

I. INTRODUCTION

A Flight Data Recorder is a replaceable computer element used in airplanes. Its task is recording pilots' inputs, electronic inputs, sensor positions and information sent to any electronic systems on the airplane [1]. Flight Data Recorder is informally called "black box". Flight Data Recorders are designed to be quite small and carefully manufactured to withstand the influence of a high speed and the heat of an extreme temperature. An example of Flight Data Recorder of a commercial airplane can be seen is Figure 1.

Up to date high density FLASH memory devices have facilitated the SSFDR (Solid State Flight data Recorder) to be manufactured with a quite larger memory size. Many airplanes are now equipped with solid-state flight data recorders and no longer make use of disk drives [2]. Additionally, in past twenty five years the density of memory chips has significantly multiplied and the capability to record thousands of parameters for hundreds of flight hours in flight data recorders or quick assess recorders is now possible.

On the other hand, unlike the debate in the personal computers area, in the embedded computing area and especially in Flight Data Recorders one and all are of the same opinion that the memory space size is too small. Actually, storage space is less than one percent of storage space available on a conventional desktop computer. Typically, In an unexceptional embedded computer system there is an electronic card with a plain processor supporting a small Solid State Device that has just about 1-4GB of memory space for all of the system files. Usually it is impossible to insert additional memory space such as Hard Disk Drive or even SD reader because of hardware constraints, system constraints, size constraints, and power consumption constraints [3].

Figure 1. Flight Data Recorder

So it is understandable why we cannot install a full operation system environment which includes a compilation chain (Tool Chain), in such a small memory space. For instance, a basic installation of Gentoo Linux distribution with a command line user interface, a stage-3 compilation tool chain, and its Portage package manager, without any graphical interface or other packages requires 1.5 GB. While installing Windows operating system takes much more memory space.

The easiest solution for this is removing features, installing only the essentials, and developing lighter applications for the embedded cards of Flight Data Recorders.

Compression algorithms are employed by the manufacturers of flight data recorders and may turn out to be even more relevant with the introduction of video flight data recorders. Although video flight data recorders are quite old [4], latest video compression techniques have a considerable compression ratio which is commonly more than some hundreds i.e. the compressed file will be much less than %1 of the original data [5]. This explains why the compression concern has been reborn even though the memory capacity is much larger nowadays.

A common difficulty is when Flight Data Recorders run out of memory space [6]. Because the designers of the flight data recorder are concerned of this lack of memory, they design the flight data recorders' memory management process to make a constant effort to reduce the used memory space [7]. A photograph of a Flight Data Recorder's storage device can be seen in Figure 2.

Figure 2. Flight Data Recorder's storage device

Though, unlike Flight Data Recorders, in many other embedded computer systems, more often than not disks are not overloaded; therefore it will be usually better to keep old versions of important files on the disks even though in most cases we will not be using the old versions [8].

Nowadays, passengers in airplanes can use Internet and cellphones [9]. The ability to wirelessly connect the network is mainly based on two techniques:

1. Cellular based network that employs many cellphone towers over ground. This method is obviously unsuitable over seas. The towers have been fabricated to direct their signals at the sky rather than along the ground. The airplanes catch the information using a receiver installed on their underside. When the information arrives at an airplane, it will be distributed throughout the cabin by the use of a conventional Wi-Fi system.
2. Satellite Internet access provided through communications satellites. Similarly to the cellular based network, when the information arrives at an airplane, it will be distributed throughout the cabin by the use of a conventional Wi-Fi system.

We suggest using this communication technique to send the information of the flight data recorder to an IaaS cloud. This way there will be no need to handle the memory storage. The burden of the memory storage handling will move out to the cloud; however, the transmitted data might be exceedingly large; therefore a compression might be needed before the sending.

II. ENABLING DATA TRANSMISSION FROM A FLIGHT DATA RECORDER TO A CLOUD

With the aim of transferring an adequate amount of data from a flight data recorder to a cloud, a compression is required to be applied. For such proposes, as with the computational codes run across today's flight data recorders, compression techniques cannot be used arbitrarily. Their use must be dynamically configured to match current requirements i. e. desired transmission rates and current platform resources i. e. network bandwidth and CPU load.

The system presented in this paper enable the flight data recorder to automatically configure the compression technique to fit the current requirements. When enough network bandwidth is available, for instance, no compression will be applied, thereby the computational loads will be reduced; however, when network bandwidth is not enough for the transmitted data, the transmitted data will be compressed. The particular compression technique will be automatically selected, using dynamic data sampling techniques to assess the effectiveness and current rapidity of compression.

The objective of this paper is guaranteeing that the rate of compression speed due to available CPU resources and the compression efficiency will create suitable data volumes transmitted over the network at rates that match current available network resources as well as application requirements.

Given whichever data sets, the tradeoffs in compression rapidity vs. reductions in required network bandwidth will be calculated. A configurable process to select a suitable compression technique based on the calculation has been developed. The technique selection process takes into account compression rapidity, current machine load, efficiency of the compression techniques, type of data and available network bandwidth. Because platform resources change, technique selection is performed frequently, throughout the lifetime of data transmitted by the flight data recorder.

III. COMPRESSION METHODS

Compression techniques reduce data size by applying compression and decompression techniques to data. This section succinctly reviews the techniques employed in this work, in order to let the reader to better understand the tradeoffs in using these different techniques.

A. *Huffman Compression*

Huffman coding [10] has been the first practical compression method. Huffman coding is usually not used in a standalone mode[11]; just within more complex compression techniques like JPEG[12,13]. The concept of Huffman coding is assigning a shorter codeword to a common item and a longer codeword to an uncommon item. The following algorithm shown in the recursive pseudo code below describes how Huffman chooses these codewords with minimum average size for items A1,...,An of lengths L1,...,Ln, where P1,...,Pn are the items' probabilities.

If (n==2) then

{0,1}

Else

Combine the 2 smallest probabilities Pn,Pn-1

Solve for P1,P2,...,Pn-2,Pn-1+Pn

If Pn-1+Pn is represented by X then

Pn-1 will be represented by X0

Pn will be represented by X1

"Solve for" means calling the function again with n-1 elements because Pn-1 and Pn become one element, as indicated by Pn-1+Pn.

The main advantages of Huffman codes are their simplicity and rapidity. These codes work well for binary data when string repetition is rare. Huffman assumes that each character has no relation to the adjacent one; therefore Huffman usually does not perform well on texts. Huffman's complexity is O(m+nlogn) where m is the size of the text and n is the size of the alphabet. It should be noted that this is much better than the Burrows-Wheeler compression technique described below, which has a complexity of O(mlogm).

B. *Arithmetic Coding*

Huffman coding makes use of a string of bits for each item in the original date. The arithmetic coding technique[14,15] improves on this by using fractions of bits as codewords, where one bit can be `owned' by several items. Therefore, a codeword can be represented by a non-integer numbers of bits (e.g., by 4.1 bits). The arithmetic coding technique is described by the following pseudocode:

Let L be a set of items.

Each item i in L has a probability Pi within [0,1], such that:

$$\sum_{i \in L} P_i = 1$$

Each item is represented by the interval:

$$[\sum_{j<i} P_j \, , \sum_{j \le i} P_j \,)$$

Repeat until EOF:

The current interval is divided into sub-intervals according to the items' probabilities.

Replace the current interval by the sub-interval of the items that were read.

Write into the compressed file the shortest binary fraction available in the current interval.

C. *Lempel-Ziv Methods*

The conventional dictionary compression technique is Lempel-Ziv coding[16]. WINZIP[17] and gzip[18] along other common practical compression tools employ versions of Lempel-Ziv coding. Whereas Huffman coding and Arithmetic coding do not consider an item's surroundings, the main advantage of Lempel-Ziv methods is that they consider previous appearances of strings. The concept of the algorithm is described herein below:

Let x1,...,xn be a sequence of items.

We want to find a sub-sequence xk,...,xm which holds:

$$P(x_k,...,x_m) > \prod_{i=k}^{m} P(x_i)$$

For example p(qu)>p(q)·p(u).

Lempel-Ziv scheme puts a pointer into the place of each previewed string. We use a version of Lempel-Ziv that compresses these pointers by Huffman coding[19]. The pointers of Lempel-Ziv look like (234,6), which means go backward 234 bytes and copy 6 characters. Most pointers point to close data, and a copy of just a small number of bytes is done, so both of the numbers have a tendency to be small. These pairs of numbers are then replaced by Huffman codewords, which give shorter representation for small numbers.

D. *The Burrows-Wheeler transformation*

The Burrows-Wheeler transformation[20] is a dictionary compression technique. This technique utilizes repetitions of words' sequences in order to improve compression. The technique is lossless i. e. no information is lost in the compression procedure. Burrows-Wheeler transformation outperforms Lempel-Ziv coding; therefore the use of Burrows-Wheeler transformation in a variety of compression utilities is widespread; however the execution time of Burrows-Wheeler transformation is normally very high.

The technique has several steps:

The first step produces pointers to all character of the data being compressed. The pointers are sorted according to the characters to which they are pointing. The preceding characters of each of the pointers are conveyed to the next step according to the order of the sorted pointers. Essentially, this sequence of characters in the output of this step has the same characters as in the original data, but the order of the characters will be different.

The second step performs a "move to front" algorithm. This algorithm keeps all 256 potential characters in a list. When a character is to be sent to the next step, its position in the list will be sent in its place. After a replacement of a character is sent, it will be moved from its current position in the list to the front of the list.

The next step applies a run-length coding to the output of the previous step. The output of the run-length coding is compressed usually by Arithmetic coding; however Huffman coding can also be applied.

The main disadvantage of the Burrows Wheeler transformation is its slow execution time, because of the need to sort the data. In order to reduce this execution time, usually the data is split into blocks, at some loss in compression efficiency, because shorter data is less effectively compressed. This paper uses the SGI version of the Burrows-Wheeler Transform[21].

In order to enable us to decompress the data when the order of blocks received does not exactly correspond to the order in which it is sent, we have adapted the Burrows-Wheeler method, as explained here:

Each data is split into block of a number of bytes. The Burrows-Wheeler Transform compresses each block. Then, blocks are processed by the move to front procedure, followed by run-length coding. The run-length coding has been changed to use a run-length of at most 254 characters, so that the 255th character never appears. Instead, the 255th character is placed at the end of each compressed block. Next, all of the blocks are compressed together using Huffman coding. Huffman can be synchronized easily, as shown in [22,23]. This indicates that if a Huffman-encoded data is read from any arbitrary point, it possibly will have a few erroneous bytes in the beginning, but the rest of the characters will be correct. So, the compressed data can be decoded from any arbitrary point, since Huffman will keep track of character positions, and when position 255 is observed, a new block has been detected.

E. Method Comparison

The attributes of these compression techniques have been evaluated; accordingly the system will be able to choose the most appropriate compression technique for any given attribute of the data, the available communication bandwidth and available CPU cycles.

Figure 3 qualitatively ranks compression techinques, scaled as 4 levels:

- Excellent
- Good
- Satisfactory
- Poor

Given these technique evaluations, the following selection algorithm chooses the compression technique most suitable for the current execution environment.

	Burrows-Wheeler	Lempel-Ziv	Arithmetic	Huffman
Compress files with string repetitions	Excellent	Excellent	Poor	Poor
Compress files with low entropy	Excellent	Poor	Excellent	Excellent
Compression Efficiency	**Excellent**	**Good**	**Poor**	**Poor**
Time of Compression	Poor	Satisfactory	Poor	Excellent
Time of Decompression	Satisfactory	Excellent	Poor	Excellent
Global Time	**Poor**	**Good**	**Poor**	**Excellent**

Figure 3. Attributes of compression techniques

In this algorithm, we use the term "reducing speed" to capture the speed at which a certain technique can compress data, given currently available CPU cycles. This speed is checked repeatedly, as subsequent blocks of data are compressed. In addition, the speed with which compressed blocks are accepted by receivers is repeatedly checked, thereby analyzing both current network bandwidth and receiver speed. These end-to-end numbers are more relevant than knowledge of actual network bandwidth, because decompression requires the use of receivers' CPU cycles.

The sizes of the blocks have been preferred based on the common page size [24,25] and the efficiency of compression techniques derived from [26]. The ratios between the sending time and the reducing speed size have been set according to the statistics detailed in Figure 6. The efficiency of the sampling has been set according to the numbers of Figure 4. Obviously, this information is specific to the particular data; However, these numbers can be easily tuned if needed by sampling even a small piece of data [27] extracted from the original data and send this piece of data over an unloaded line employing unloaded CPUs. It should be noted that usually the numbers being used are very close to the constants details here, so we put these constants to give the reader an impression what the scope of the numbers is.

Assume the reducing size speed of first block is infinity.

While not EOF

Take a block of 128KB.

If (sending time) > 0.83*(the reducing size speed of Lempel-Ziv)

If sampling has been compressed into less than 48.78%

If (sending time) > 3.48*(the reducing size speed of Lempel-Ziv)

Use Burrows-Wheeler

Else

Use Lempel-Ziv

Else

Use Huffman

Else

Do not Compress

Fork a sampling process to compress the first page (4KB) of the next block by Lempel-Ziv and use its output to decide on the reducing speed size and the compression ratio for the next 128KB block.

Send the block.

Wait for child process.

IV. Experimental Results

Data flight recorders generate various information; therefore the compression techniques used in this paper i. e. Huffman, Arithmetic, Lempel-Ziv and Burrows-Wheeler, have been tested with multiple datasets, including a binary dataset and a text dataset. For the text dataset, the compression ratios are shown in Figure 4.

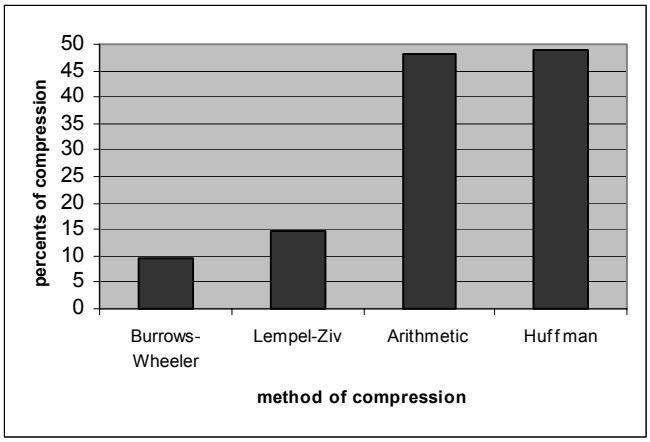

Figure 4. Compression ratios

Decisions about suitable compression techniques should be based not only on data sizes or link speeds, but also on data characteristics. Huffman codes and Arithmetic codes are suitable for low entropy data, while Lempel-Ziv methods are good at handling data with string repetitions. Burrows-Wheeler handles both of these cases.

The consequent approach taken in our work is one that samples data as it is being produced and transported, to detect whether data has low entropy, string repetitions, or both. The results of such sampling are used to choose a suitable compression method.

The crucial statistics from these experiments is the speed with which a CPU compresses some large amount of data. Figure 5 summarizes the test results accomplished with two flight data recorders. Because of confidential commercial issues, we cannot detail the names of the flight data recorders and we will refer to them simply as "New FDR" and "Old FDR".

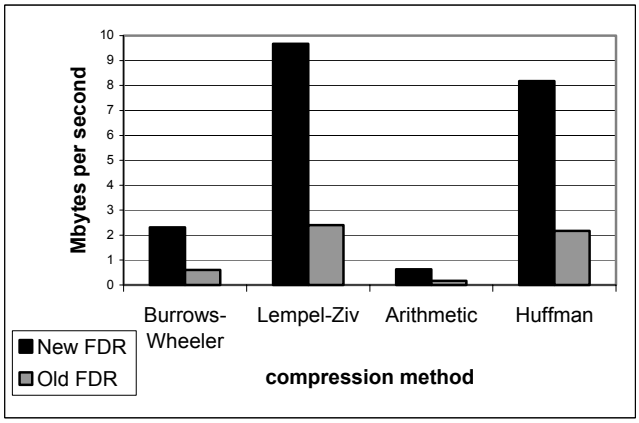

Figure 5. Reducing size speed

Figure 5 shows what is called in this paper the "reducing size speed" of different flight data recorder's processors, which is the ability of a flight data recorder's processor to reduce amount of bytes per second. If such a memory space reduction can be performed faster, than the transfer time for a given amount of data, it is worth (time-wise) to compress the data.

If a flight data recorder's processor is fast, but the communication line is slow, even a multipart compression method can be chosen. In the opposite case, i. e. the CPU is slow and the communication line is fast, no compression technique will be assigned. Between those two extremes, the use of a fast and uncomplicated compression method will be chosen.

Usually cellphone lines and Internet lines have a large standard deviation of the transfer speed. This brings about different compression technique selection at different points of time.

The conclusion from the above figures is that if the CPU is very fast, then Burrows-Wheeler will be the best technique. Burrows-Wheeler has a poor ratio of reducing MBits per second, but if the CPU is fast enough, the CPU will be able to pay back any lost time.

If the communication line is very fast, Huffman will be the best technique. Huffman has a poor compression ratio, but it compresses data quite fast.

When an intermediate case occurs, Lempel-ZIV seems to be a good compromise between compression time and transfer time.

Arithmetic coding does not appear useful for the group of applications considered in this paper. Compression time of the Arithmetic Coding appears too long ad the compression ratio is too similar to Huffman coding. This suggests that the Arithmetic coding should not be used at all.

Figure 6 show the switching of compression techniques over time. The data being compressed, transported, and decompressed is a set of information captured from a large company. This data set has a high rate of data repetitions, so the best techniques to be used were Lempel-Ziv and Burrows-Wheeler.

These comments explain the automatic decision-making depicted in Figure 6. Initially, with no network load, no compression is performed (labeled as `1' in the figure). With increasing network load, the first compression method used is Lempel-Ziv (see `2' in the figure), followed by Burrows-Wheeler (see `3') under high network loads.

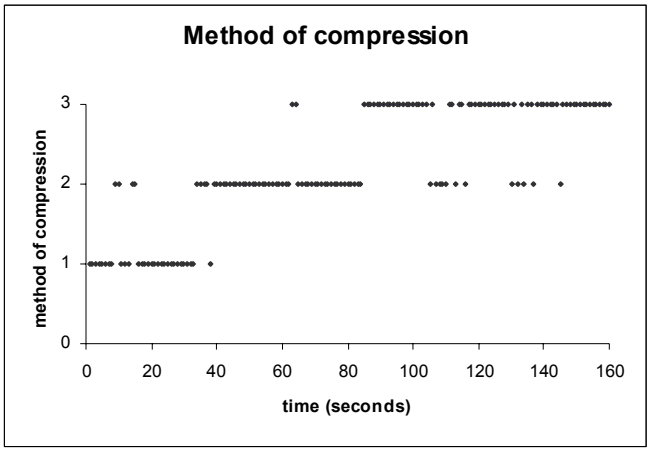

Figure 6. Switching of compression techniques

Figure 6 depicts the selection of compression technique over time; whereas Figures 7 and 8 show the compression times and the sizes of the compressed blocks, respectively, achieved by these techniques. These figures clearly show that the relatively small improvement in data reduction achieved by the using of Burrows-Wheeler justify its usage only under very high network loads.

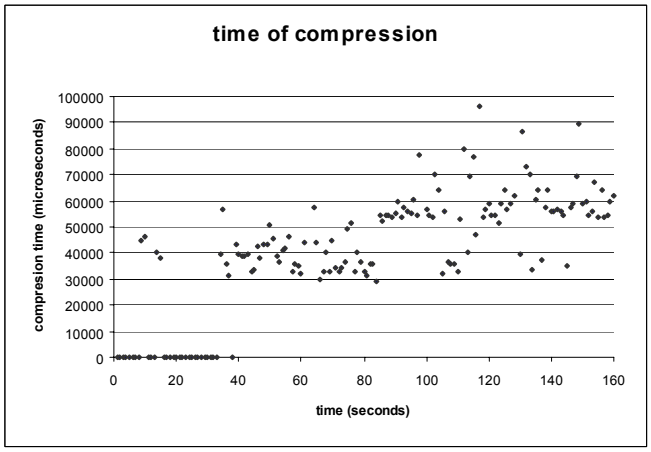

Figure 7. CPU time compression

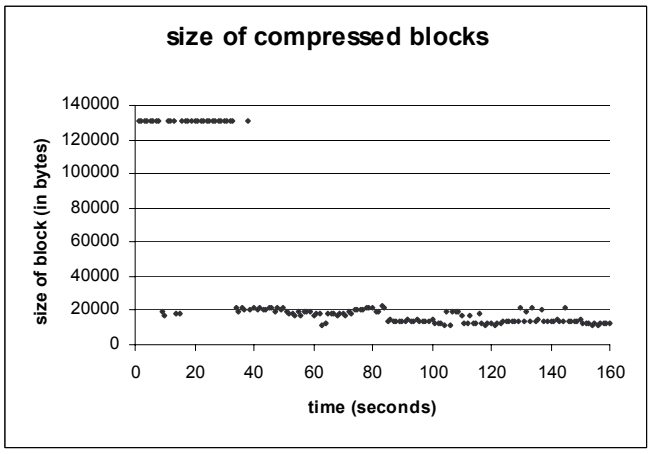

Figure 8. Transmitted compressed blocks' size

V. CONCLUSIONS

Signals and data generated by data flight recorders are commonly kept on an internal memory device like hard disks or FLASH memory devices, but the internal memory device of data flight recorders is typically small [28,29]. In addition Flight Data Recorders just write internal memory device and almost never read the disks; therefore a compression can be beneficial for such systems. The suggested data flight recorder suggests a way of using a cloud as the memory storage device instead of the internal memory device. A compressed data system will make sure the transmitted data will be reduced, while making sure the other tasks will have almost no reduction in performance. The compression algorithm (whichever algorithm will be chosen by the system) will only use the available CPU cycles.

Results are encouraging. The transmission system is able to select the compression algorithm intensity and whether to compress or not at real time based on available CPU cycles and the load on the network line, so the cloud can be a replacement for the internal memory device.

Additionally, clouds are crash-proof. The possibility of a flight data recorder to be damaged in a crash is quite small [30], because flight data recorders are designed to sustain even very severe crashes; however if an airplane is crashed, the cloud will not be there, so the potential damage to the cloud is absolutely zero.

REFERENCES

[1] Li, Chang You, Gui Rong Ye, and Quan Fa Yang. "Design of Flight Data Signal Generator System." In Applied Mechanics and Materials, vol. 556, pp. 5143-5147. 2014.

[2] Yair Wiseman and Alon Barkai, Smaller Flight Data Recorders, Journal of Aviation Technology and Engineering, Vol. 2(2), pp. 45-55, 2013 .

[3] Yaghmour K., Masters J., Gerum P. and Ben-Yossef G., Building embedded linux systems, O'Reilly Media, Inc., 2008.

[4] Armstrong, Herbert B. "Improving aviation accident research through the use of video." ACM SIGCHI Bulletin, Vol. 21, no. 2, pp. 54-56, 1989.

[5] Horowitz, M., Kossentini, F., Mahdi, N., Xu, S., Guermazi, H., Tmar, H., Li, B., Sullivan, G.. J. and Xu, J., "Informal subjective quality comparison of video compression performance of the HEVC and H. 264/MPEG-4 AVC standards for low-delay applications", In Proc. Applications of Digital Image Processing XXXV, San Diego, California, USA p. 84990W, August 12, 2012.

[6] Pinchas Weisberg and Yair Wiseman, "Efficient Memory Control for Avionics and Embedded Systems", International Journal of Embedded Systems, Vol. 5(4), pp. 225-238, 2013 .

[7] Wu, J. C., Banachowski, S. and Brandt, S. A., "Hierarchical disk sharing for multimedia systems", In Proc. of the international workshop on Network and operating systems support for digital audio and video, pp. 189-194, 2005.

[8] Muniswamy-Reddy, K., Wright, C. P., Himmer, A. and Zadok, E., "A Versatile and User-Oriented Versioning File System", In Proc. of the Third USENIX Conference on File and Storage Technologies (FAST 2004), San Francisco, California, pp. 115-128, March 31-April 2, 2004.

[9] Sakhaee, Ehssan, and Abbas Jamalipour. "The global in-flight internet." Selected Areas in Communications, IEEE Journal on 24, no. 9, pp. 1748-1757,2006.

[10] Huffman D., "A method for the Construction of Minimum Redundancy Codes", In Proc. of the IRE 40, pp.1098-1101 1952.

[11] Dandekar, Onkar. "Full-Band CQI Feedback by Huffman Compression in 3GPP LTE Systems", International Journal of Computer Applications, Vol. 77, no. 9, pp. 37-42, 2013.

[12] Wallace G. K. "The JPEG Still Picture Compression Standard", Communication of the ACM 34, pp. 3-44, 1991.

[13] Yair Wiseman, "The still image lossy compression standard – JPEG", Encyclopedia of Information and Science Technology, Third Edition, Vol. 1, Chapter 28, 2014.

[14] Howard P. G. and Vitter J. S., Arithmetic Coding for Data Compression, Proceedings of the IEEE, Vil. 82, No. 6, pp. 857-865, 1994.

[15] Yair Wiseman, A Pipeline Chip for Quasi Arithmetic Coding, IEICE Journal - Trans. Fundamentals, Tokyo, Japan, Vol. E84-A No.4, pp. 1034-1041, 2001.

[16] Yair Wiseman, The Relative Efficiency of LZW and LZSS, Data Science Journal, Vol. 6, pp. 1-6, 2007.

[17] WinZip, Nico Mak Computing, Inc., Mansfield, CT, USA 1998.

[18] WinZip, Nico Mak Computing, Inc., Mansfield, CT, USA 1998.

[19] Peterson, Peter Andrew Harrington, "Datacomp: Locally-independent Adaptive Compression for Real-World Systems", PhD Thesis, UCLA Electronic Theses and Dissertations, 2013.

[20] Burrows M. and Wheeler D. Block sorting Lossless Data Compression Algorithm, System research center, research report 124, Digital System research Center, Palo Alto, CA 1994.

[21] Yair Wiseman, Burrows-Wheeler Based JPEG, Data Science Journal, Vol. 6, pp. 19-27, 2007.

[22] Shmuel T. Klein and Yair Wiseman, "Parallel Huffman Decoding with Applications to JPEG Files", The Computer Journal, Oxford University Press, Swindon, UK, Vol. 46(5), pp. 487-497, 2003.

[23] Shmuel T. Klein and Yair Wiseman, "Parallel Huffman Decoding", Proc. Data Compression Conference DCC-2000, Snowbird, Utah, USA, pp. 383-392, 2000.

[24] Pinchas Weisberg and Yair Wiseman, "Using 4KB Page Size for Virtual Memory is Obsolete", Proc. IEEE Conference on Information Reuse and Integration (IEEE IRI-2009), Las Vegas, Nevada, pp. 262-265, 2009.

[25] Moshe Itshak and Yair Wiseman, "AMSQM: Adaptive Multiple SuperPage Queue Management", Special issue of the International Journal of Information and Decision Sciences (IJIDS) on the best papers of IEEE Conference on Information Reuse and Integration (IEEE IRI-2008), Vol. 1, No. 3, pp. 323-341, 2009.

[26] Shmuel T. Klein and Yair Wiseman, Parallel Lempel Ziv Coding, Journal of Discrete Applied Mathematics, Vol. 146(2), pp. 180-191, 2005.

[27] Yair Wiseman, Karsten Schwan and Patrick Widener, "Efficient End to End Data Exchange Using Configurable Compression", Operating Systems Review, Vol. 39(3), pp. 4-23, 2005.

[28] Yang, L., Dick, R. P., Lekatsas, H., and Chakradhar, S., "CRAMES: Compressed RAM for embedded systems", In Proc. International Conference on Hardware/Software Codesign and System Synthesis, pp.93-98, 2005.

[29] Xu, X. H., Clarke, C. T., and Jones, S. R., "High performance code compression architecture for the embedded ARM/Thumb processor", In Proc. Conf. Computing Frontiers. pp. 451-456, 2004.

[30] Berecz, Endre, and Michael Winterhalter, "Crash survivable memory unit", U.S. Patent 8,121,752, issued February 21, 2012. J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.