Lightweight Packing of Log Files for Improved Compression in Mobile Tactical Networks

Peter Mell
National Institute of Standards and Technology
Gaithersburg, MD
peter.mell@nist.gov

Richard E Harang
U.S. Army Research Laboratory
Adelphi, MD
richard.e.harang.civ@mail.mil

ABSTRACT

Devices in mobile tactical edge networks are often resource constrained due to their lightweight and mobile nature, and often have limited access to bandwidth. In order to maintain situational awareness in the cyber domain, security logs from these devices must be transmitted to command and control sites. We present a lightweight packing step that takes advantage of the restricted semantics and regular format of certain kinds of log files to render them substantially more amenable to compression with standard algorithms (especially Lempel-Ziv variants). We demonstrate that we can reduce compressed file sizes to as little as 21% of that of the maximally compressed file without packing. We can also reduce overall compression times up to 64% in our data sets. Our packing step permits lossless transmission of larger log files across the same network transmission medium, as well as permitting existing sets of logs to be transmitted within smaller network availability windows.

Keywords
security, logs, compression, Lempel-Ziv

1. INTRODUCTION

Compression algorithms are typically designed to be general purpose; encoding arbitrary binary sequences into more compact representations by removing any inherent redundancy in the message. As such, they typically do not take advantage of specific features of some data formats that might enable more efficient compression. In contrast to general binary formats, many information technology (IT) logs allow for significant syntactic modifications without loss of semantics. In this work, we identify lightweight lossless syntactical packing that can be applied to security logs to make them more amenable to compression.

We focus on packing that will optimize the Lempel-Ziv Markov Chain Algorithm (LZMA); a widely used and effective lossless file compression algorithm, which is (anecdotally if not formally) known to be highly effective at compressing structured data [1]. Some version of the LZ77 algorithm [2], from which LZMA is derived [1], is used by default on many Unix and Windows systems and is included in many file compression suites [3] [4].

For our approach to apply, the data must conform to two pre-requisites often existent within IT logs and database tables. The data must be either explicitly tabular, or capable of being converted to one or more tabular representations. Also, the rows of each table must contain ordering identifiers (e.g., time stamps) or be atomic in nature, allowing for reordering without loss of semantics. We exploit these properties to construct collections of row sorted matrices that are then transposed and serialized prior to compression. While both explicit and implicit transposition of matrices has seen historical use in the area of statistical databases for both compression and speed of access [5] [6], this approach appears to have seen limited application to the area of general purpose log compression (though see related work, below).

We demonstrate that our general-purpose packing approach allows us to reduce maximally compressed file sizes to as little as 21% of that attainable without packing. At higher compression levels, our approach also significantly reduces overall compression times, up to 64% in our data sets.

While enhanced file compression itself is a generally useful capability, there are four IT security scenarios in which our approach may be of substantial assistance. 1) In mobile tactical edge networks and emergency response networks, bandwidth may be low or of extreme value and thus minimizing throughput of traffic is critical. 2) Centralizing security logs from distributed cloud applications geographically dispersed across multiple data centers (such data movement is typically expensive and slow). 3) Centralizing security logs from all IT devices on an enterprise network for correlation analysis. 4) Archiving enterprise wide security logs for long term storage.

We test our approach by applying LZMA compression to Windows security logs, Snort [7] intrusion detection system alerts, data from the National Software Reference Library (NSRL) [8], and a publically available set of log data (the "fighter-planes.com" HTTP log data [9])[1]. We then compare the results to applying LZMA on the same data after using

[1] Any mention of commercial products or reference to commercial organizations is for information only; it does not imply recommendation or endorsement by the U.S. government nor does it imply that the products mentioned are necessarily the best available for the purpose.

our data packing techniques. We also apply other popular compression algorithms for comparative purposes.

2. RELATED WORK

The work of both [10] and [11] have similar goals to ours, focusing on pre-processing log files in preparation for further treatment by a general-purpose compression algorithm. The work of [10] is general purpose, like ours, and exploits the line-to-line similarity of log files by use of back references. The work of [11] focuses exclusively on Apache HTTP logs and utilizes the same 'transposition' step that we exploit. Both works make use of the fighter-pilot HTTP log data set [9] (hereafter denoted "FP") as one test case, enabling direct comparisons. We compare the results with respect to packing followed by compression with the LZMA algorithm in Table 1. Our method outperforms the general purpose method of [10] on the FP data set while underperforming the application specific optimized method of [11]. While the work of [11] provides a higher compression ration than our own, their method (as they note) is not extensible to other formats, as they use several optimizations specific to web logs.

Table 1 - Comparison to related work, bits per character (bpc)

Approach	LZMA	Packing+LZMA
Present work	0.323 bpc	0.186 bpc
Grabowski and Deorowicz [11]	0.360 bpc	0.129 bpc
Skibinski and Swacha [10]	0.317 bpc	0.246 bpc

Other related work in [12] also discusses the compression of log files via transformation to matrices; however they focus on lossy compression via the use of numerical encoding followed by factorization to sparse matrices.

3. DATA PRE-REQUISITES

Our approach for packing data applies only to data with certain properties. The data to be packed must consist of a sequence of entries, which must all either have the same number of fields (referred to as columns) or have some column (denoted a 'key' field) such that all entries with the same key have the same number of columns. The data can thus be treated as either a single matrix or converted through use of the key into a collection of matrices. In addition, the rows must either explicitly contain ordering information (e.g., a timestamp or sequence identifier) to enable reconstruction of the original ordering of the matrix, or any given ordering of the rows must be equivalent (i.e., the ordering of the rows must have no semantic significance). These properties are exhibited by many, but certainly not all, security logs.

4. OVERVIEW OF LEMPEL-ZIV COMPRESSION

Two variants of Lempel-Ziv compression were originally proposed; the LZ78 algorithm [13] (later extended into the more popular LZW algorithm [14]) which constructs a dictionary of observed sequences in an on-line fashion, and the LZ77 algorithm [2] which treats some fixed number of recently observed uncompressed bytes in the input stream (the "sliding window") as a dictionary. The LZMA algorithm – discussed in more detail below – is derived primarily from the LZ77 algorithm.

The original implementation of LZ77 searches the sliding window for prefixes to the current substring, and uses fixed-width encoding to indicate the starting position, the number of bytes to replicate, and the single-character suffix to the string. Practical implementations using the LZ77 algorithm (such as DEFLATE [15]) typically incorporate other modifications such as Huffman coding.

The LZMA algorithm, despite being an open-source implementation [1], is not well documented, and formal descriptions of the algorithm (outside of source code) do not appear to be available. LZMA is based on the LZ77 [2] algorithm, but contains a number of modifications (including special tokenization, look-aheads, and binary range coders) to improve compression efficiency. While the details of the different LZ77 based implementations are complex, the fundamental process of encoding the current portion of the stream in terms of some previously observed portion remains consistent. This forms the basis for the optimization which we discuss below.

5. DATA PACKING METHOD

Our data packing method enables the LZMA sliding window greater opportunity for reuse of recently processed data. While the design goal is similar to that of the Burrows-Wheeler transform [16] used in the BZip2 algorithm, that is, placing similar data close together to enhance the ability of other algorithms to compress it, our approach is significantly simpler both in concept and implementation. Our approach is shown in Figure 1.

1. Read in a log file or sets of log files
2. Add each log entry to a minimal set of matrices with particular properties (see text)
3. Sort each matrix
4. Transpose each matrix
5. Compose all matrices into a single data structure by creating a dictionary in which all matrices are stored by hashing a unique field name
6. Serialize the dictionary to disk using a general purpose serialization routine
7. Compress with a standard compression algorithm

Figure 1. Packing Steps

Packing occurs in seven steps. In step 1, log data is imported. In step 2, each log entry is added to a set of matrices such that the number of columns of the matrix match the number of fields in the entry, and the semantics of each matrix column and entry field match. If no such matrix is available for a particular log entry, a new matrix is created to hold that entry. This operation can be implemented in O(n) time and space with respect to the number of entries in the data.

In step 3, once the structure of the data has been normalized to a collection of matrices in which each row corresponds to an entry and each column to a field, we permute the rows of these matrices by sorting based upon the value of the first column, breaking ties by the value of columns to the right if required. Such a sorting operation is of worst-case time complexity O($n$log$n$) and space complexity O(n), however note that by subdividing the data based on the key field as described above, the overall time complexity may be reduced substantially. This occurs in some of our data sets where the number of matrices formed is a function of the size of the input set.

In step 4 we transpose the sorted matrices, such that the first column of each matrix now becomes the first row. Matrix transposition is of time and space complexity O(n) as well.

In step 5, we store each group in a dictionary, keyed using the key field value associated with each matrix. This dictionary is serialized to disk using a general purpose serialization routine in step 6. The serialized dictionary is then compressed in step 7 using a standard compression algorithm. We may have been able to achieve better compression by creating a custom serialization for each log type (similar to the customizations used in [11]), however such fine-tuning limits the generality of the approach. Serialization is O(n) in time and space complexity, while the complexity of the final compression step will be dependent on the algorithm selected.

The second step of packing allows the data to be coerced into a matrix form so that the transposition operation of the fourth step may be performed. The combination of the sorting in the third step and the transposition of the fourth step places similar data adjacent in memory, allowing for a greater degree of matching within the sliding window of the LZ-based compressor, and thus a higher degree of compression. In addition, as the columns representing the same type of data will often have more similar formats and content than different columns in the same record, the transposition allows for significant improvement even when the sorting step does not penetrate significantly to those columns in the pre-transposed matrix.

We also note that the order of the columns within each particular matrix plays a role in the effectiveness of the sort operation. We tried reordering the columns by increasing and decreasing entropy but found no consistent approach to apply to optimally order the columns for each group within a log type. It is not clear that this is a tractable approach because the algorithm to reorder the columns may take significant computational complexity. Our entropy ordering experiments often more than doubled the overall compression time. It would also require storing the column reordering information within the final compressed file. We leave such an approach and the possible discovery of an optimal column reordering algorithm to future work.

6. EXPERIMENTAL DESIGN

We evaluate our approach through packing and compressing data in 4 distinct file formats:

1. Microsoft Windows security logs collected in Windows Event Viewer (.evtx) format from 11 Windows 7 workstations in a large network over a 6 month period in 2013, comprising approximately 1.62 million records.
2. Snort [7] network intrusion detection system (NIDS) alerts from an enterprise-scale production network, collected approximately 2.25 million alerts over a period of 27 days in 2012 using a combination of the Snort Emerging Threats (ET) and Vulnerability Research Team (VRT) rule sets.
3. National Software Reference Library (NSRL) [8] data, primarily used to whitelist files during computer forensic examinations. This data was subsampled to the first 3 million records and serialized from the January 2014 'unique' data file, excluding file hash values (which are large and incompressible thus inhibiting compression algorithm comparison). We also examined smaller subsets to evaluate the impact of source data size on packing effectiveness.
4. Publically available HTTP server log data (attributed to the "fighter-planes.com" website) recording a series of HTTP requests to the server between the hours of 03:07 and 19:14 on February 3, 2003. The data includes IP address, date/timestamps, the HTTP request type and URI, response codes, referrers, and user agent information for 109,481 distinct requests.

For compression, we used the open source 7-Zip software [17] running on a commodity computer with a quad-core processor and 8GB of memory. With respect to compression algorithms, we performed experiments using all implemented algorithms: LZMA, LZMA2, PPMD, BZip2 (see the 7-Zip documentation for algorithm details). 7-Zip offers 5 levels of compression for each algorithm (levels 1, 3, 5, 7, and 9). While this does not represent a resource-constrained environment, it does provide a common baseline for comparison between all reference implementations we examined. Preliminary experimentation (data not shown due to space limitations) on a Raspberry Pi microcomputer indicates that the relative performance between the approaches is consistent with the results we report here, though absolute performance clearly is degraded.

For serializing our packed data, we selected the general purpose serializer MessagePack [18] due to its speed of serialization and relatively low size overhead compared to other standard serialization libraries such as Pickle or JSON.

7. RESULTS

We now provide empirical results for our experiments in terms of compression effectiveness and execution time. Our packing approach reliably increased compression effectiveness for the LZMA and LZMA2 methods at all levels of compression and reduced overall compression time at the higher 7-Zip compression levels (7 and 9). Results for the non LZ-based compression methods were more variable but often very good, somewhat surprising as our packing approach was not designed to support those compression algorithms.

7.1 Compression Effectiveness

Figures 1 through 4 display the various results of our compression experiments. In figure 2 we focus on the effect of packing and LZMA compression level on overall compression effectiveness relative to the compressed size of the unpacked files. Regardless of data set or compression level, the ratio of the packed and compressed file size to the unpacked and compressed file size remains relatively constant for each data set. In the best case, packing the Snort logs produced a final file only 21% the size of the compressed unpacked file at compression level 1. In the worst case, packing the FP data the final file was 66% the size of the compressed unpacked file at compression level 3.

Plots analogous to Figure 2 for other compression algorithms are not shown due to space limitations. LZMA2 results were almost identical to Figure 2. PPMd and BZip2 showed increased compression (up to a 79% reduction in final file size) using our packing approach, except for PPMd compression of the FP data at levels 1 and 3 (where our approach increased the final file size by 2.7%).

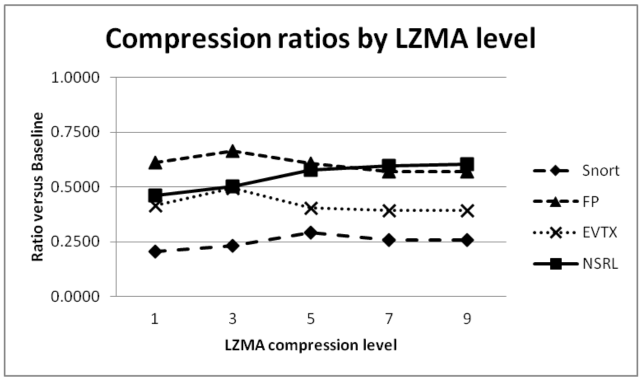
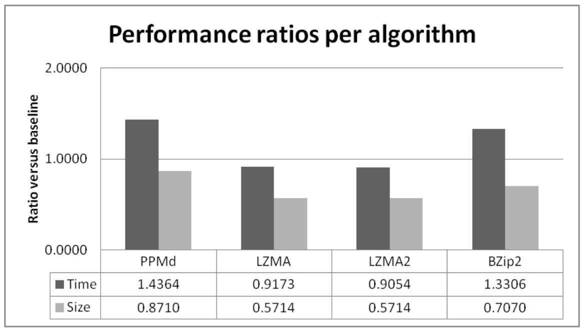


Figure 2 - Ratio of packed compressed file size to baseline compressed file size

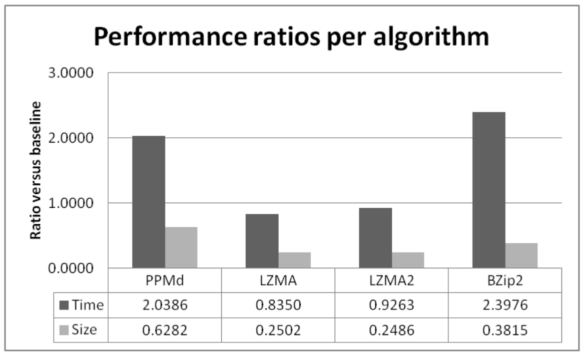
For the remainder of our compression results, we focus on the highest compression level. In Figures 3, 4, 5, and 6 we examine each data set individually at the highest 7-Zip compression level (9) with all available compression algorithms. All other settings were left at default values.

In each plot, both total compression time and final file size are presented relative to the time and file size information for compressed files without packing. Results of 1.0 for time and size indicate that the packed and compressed file was the same size as the compressed file and required the same amount of time in total (including both packing and compression) to process, for no advantage. Values less than 1.0 for size indicate that the packed and compressed file was smaller than the unpacked, compressed file; less than 1.0 for time indicate that the end-to-end processing, including packing and compression, required less time in total than straightforward compression.



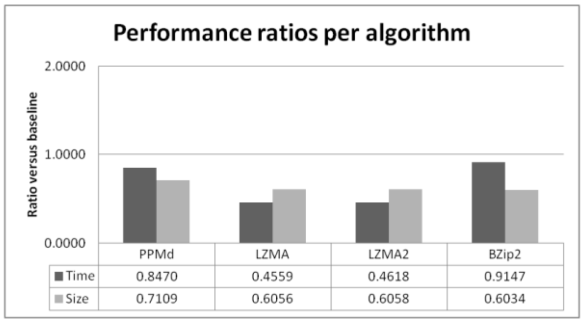
	PPMd	LZMA	LZMA2	BZip2
■ Time	1.4364	0.9173	0.9054	1.3306
■ Size	0.8710	0.5714	0.5714	0.7070

Figure 3-"Fighter Pilot" HTTP server log results



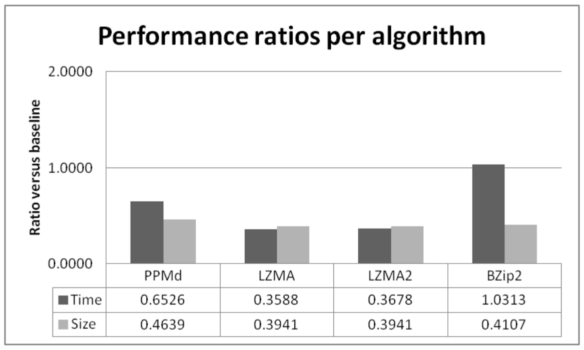
	PPMd	LZMA	LZMA2	BZip2
■ Time	2.0386	0.8350	0.9263	2.3976
■ Size	0.6282	0.2502	0.2486	0.3815

Figure 4 - Snort alert results



	PPMd	LZMA	LZMA2	BZip2
■ Time	0.8470	0.4559	0.4618	0.9147
■ Size	0.7109	0.6056	0.6058	0.6034

Figure 5 - NSRL results



	PPMd	LZMA	LZMA2	BZip2
■ Time	0.6526	0.3588	0.3678	1.0313
■ Size	0.4639	0.3941	0.3941	0.4107

Figure 6 - Windows EVTX log results

While the packing step produced an improvement with respect to file size for all compression algorithms, we obtain the best results with the LZ-based methods of LMZA and

LZMA2. As BZip2 uses several stages of preprocessing, including run length encoding, the Burroughs-Wheeler transform, and move to front encoding, the latter two of which serve primarily to reorder the data into a more compressible format prior to applying Huffman coding, it is likely that our packing is partially redundant, resulting in increased processing times without benefits as substantial as those obtained by the LZ-based algorithms. While implementation details vary, PPMd based compression constructs adaptive contextual probabilistic models for upcoming symbols in the data stream, and uses this prediction to achieve high compression rates. Due to the adaptive probabilistic model that PPMd uses for compression, the reorganization of the column entries to be adjacent in memory likely assists with the stability of the codebook, however the relevant context for successor symbols will be largely unchanged between packed and unpacked data except at entry boundaries; this likely accounts for our weaker results using the PPMd algorithm.

7.2 Timing Results

Our packing approach co-locates similar data, enabling the LZMA algorithm to reuse more data and thus execute faster, often sufficiently fast to overcome the extra time required by packing. These results are displayed in Figure 7, where we compare the ratio of compression time for packed and unpacked data for the 5 possible LZMA compression levels. In all cases, the total compression process, including packing, requires less time for packed than unpacked data at levels 7 and 9. Performance at lower compression levels appears to be substantially impacted by the sorting operation. Note the relatively high performance of NSRL data despite the large number of records. In this case, the 3 million NSRL records were segmented into 8419 separately sorted sub-tables which reduced the total cost of the sorting operation significantly. In the best case, packing EVTX data reduced the overall level 9 compression time by 64%. In the worst case, packing Snort data increased overall level 1 compression time by 228%.

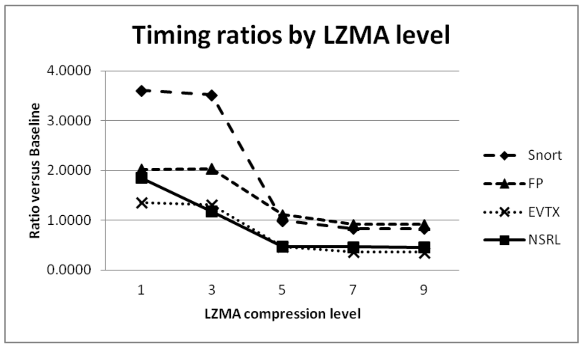


Figure 7 - timing results for LZMA compression relative to unpacked data

Additional timing information for the other algorithms at compression level 9 is also included in figures 3, 4, 5, and 6, comparing the impact of our packing scheme against unpacked compression.

The results and complexity analysis suggest that the main bottleneck in the packing step is the $O(n\log n)$ sort operation, and so the selection of a 'key' value may also be optimized to fragment the total data as effectively as possible.

While we defer detailed examination of such possibilities to future work, we do note that the compression time and effectiveness had a variable relationship. For Snort data, as shown in Figure 8, while we selected column 8 as our key to subdivide the data into matrices, this was based entirely on compression size considerations. The selection of key 3 would have allowed a small tradeoff of compression effectiveness for speed.

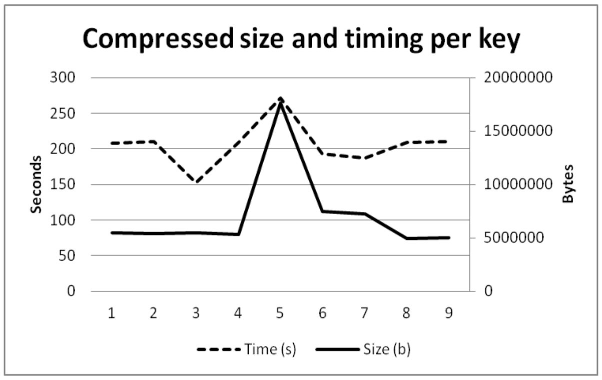


Figure 8 - Effect of key selection on Snort data

7.3 Effect of file size

To investigate the impact of file size on the compression effectiveness and timing for packed versus unpacked files, we used the NSRL data set to examine increasing input sizes. We sampled an increasing number of records, in 500,000 record increments, and compressed both unpacked and packed files using the LZMA method at a compression level of 9 (results in Figure 9). As in all other trials, timing results for packed data include all stages of packing, serialization, and compression. It is worth remarking on the fact that – despite the worst-case performance of our packing being greater than $O(n)$ – the combination of packing and compression is empirically remarkably linear within the range we examine (a simple linear best-fit line suggests a slope of 15.42 seconds per million records with $R^2 = 0.9864$, with the intercept forced to 0). Regardless of source file size, total required processing time for the packed file is approximately one-half that of the unpacked file, while the ratio of packed and compressed file size to compressed file size without packing slowly decreases from 0.7032 at 0.5 million records to 0.6056 at 3 million records, indicating an increasing advantage with respect to size from packing as larger data sets are compressed.

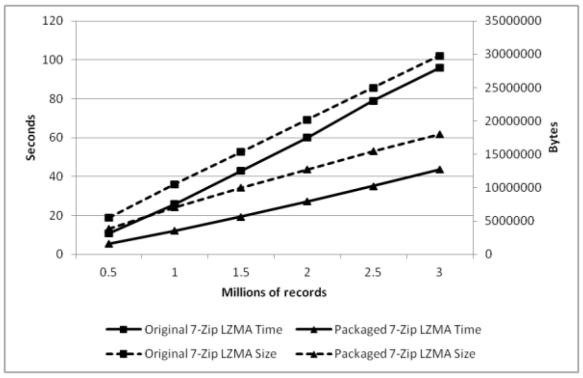


Figure 9 - Effect of variable source file size on packed vs unpacked files

8. CONCLUSIONS

We have presented a method for the lossless packing of log files that greatly enhances LZ-based algorithm compression effectiveness. Despite the overhead of packing, the approach can actually reduce overall compression time at higher compression levels.

This method leverages the fact that many kinds of logs may be converted into a matrix, which then allows sorting and transposition. We have generalized this operation to allow it to be performed on log files which may be considered a mixture of tables, possibly with a varying number of columns, and demonstrated our results on a range of formats of different sizes. We show that using LZMA compression, our packing method allows us to reliably obtain better compressed files, from 21% to 66% the size of files treated only with straightforward compression. In addition, our method is capable of accelerating the compression process at higher compression levels to sufficiently outweigh the time required to preprocess them, thus reducing the overall time to compress the file, sometimes significantly (up to a 64% reduction). This method is thus well-suited to resource-constrained networks such as tactical networks, in which transmission windows for log files may be short and difficult to anticipate and available bandwidth may be limited, but nevertheless entire log files must be transmitted. In future work, we plan to examine in more detail optimal methods for key selection, as well as the effect of permuting columns of the data prior to sorting.

9. ACKNOWLEDGMENTS

This research was sponsored by the U.S. Army Research Labs and the National Institute of Standards and Technology (NIST). It was partially accomplished under Army Contract Number W911QX-07-F-0023. The views and conclusions contained in this document are those of the authors, and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory, NIST, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes, notwithstanding any copyright notation hereon.

10. REFERENCES

[1] I. Pavlov, "LZMA SDK (Software Development Kit)," 2013. [Online]. Available: http://www.7-zip.org/sdk.html. [Accessed 15 Jan 2014].

[2] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE Transactions on Information Theory* , vol. 23, no. 3, pp. 337-343, 1977.

[3] P. Deutsch, *RFP 1952: "GZIP file format specification version 4.3"*, 1996.

[4] A. Roshal, "WinRAR archiver, a powerful tool to process RAR and ZIP files," win.rar GmbH, 2014. [Online]. Available: http://www.rarlab.com/. [Accessed 14 Jan 2014].

[5] H. K. Wong and J. Z. Li, "Transposition algorithms on very large compressed databases," in *12th International Conference on Very Large Data Bases*, 1986.

[6] H. K. Wong, H.-F. Liu, F. Olken, D. Rotem and L. Wong, "Bit Transposed Files," in *VLDB*, 1985.

[7] M. Roesch, "Snort -- lightweight intrusion detection for networks," *Proceedings of the 13th USENIX conference on System administration,* pp. 229--238, 1999.

[8] "National Software Reference Library," [Online]. Available: http://www.nsrl.nist.gov. [Accessed 17 1 2014].

[9] W. Bergmans, "Maximum Compression (lossless data compression software)," [Online]. Available: http://www.maximumcompression.com/. [Accessed 17 January 2014].

[10] P. Skibinski and J. Swacha, "Fast and efficient log file compression," in *Proceedings of 11th East-European Conference on Advances in Databases and Information Systems*, 2007.

[11] S. Grabowski and a. S. Deorowicz, "Web log compression," Institute of Computer Science, Silesian Technical University, Gliwice, Poland, 2007.

[12] G. Aceto, A. Botta, A. Pescapé and a. C. Westphal, "An efficient storage technique for network monitoring data," in *IEEE International Workshop on Measurements and Networking*, 2011.

[13] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Transactions on Information Theory,* vol. 24, no. 5, pp. 530-536, 1978.

[14] T. A. Welch, "A technique for high-performance data compression.," *Computer,* vol. 17, no. 6, pp. 8-19, 1984.

[15] P. Deutsch, *RFP 1951: "DEFLATE Compressed Data Format Specification version 1.3"*, 1996.

[16] M. Burrows and D. J. Wheeler, "A block-sorting lossless data compression algorithm.," Digital Systems Research Center, Palo Alto, CA, 1994.

[17] I. Pavlov, "7-zip," 2013. [Online]. Available: http://www.7-zip.org/. [Accessed 2013 15 Jan].

[18] S. Furuhashi, "MessagePack," 2013. [Online]. Available: http://msgpack.org/. [Accessed 2014 14 Jan].