Summary Review Documentation for

“RasterZip: Compressing Network Monitoring Data with Support for Partial Decompression”

Authors: F. Fusco, M. Vlachos and X. Dimitropoulos

Reviewer #1

Summary: This paper introduces a method for compressing network traffic data that leverages redundancy among network traffic measurement data. It introduces a different scanning method, so called Rasterzip, and uses RLE for compression. They also introduce a fast decompression algorithm with the capability for partial data decompression.

While the idea is interesting and results shown are promising, the paper completely ignores failures. It fails to compare the robustness of Rasterzip against other methods. Not sure how applicable it is in real network environment.

Strengths: The idea is promising - results show the algorithm can increase compression ratio & decompression method is fast.

Weaknesses: The paper completely ignores failures, i.e., how robust is this compression/decompression method against loss due to noise, erasure and failures in network, hardware or software? Insufficient evaluations.

Comments to authors: Overall, I find the motivation for the work lacking. Storage cost will continue to fall. One would think compressing monitoring data for later ‘transmission’ makes a more propelling case than compressing the data for storage in one place. If this is the case, then it’s absolutely crucial to consider failure/loss, which is totally ignored in this paper.

What is the robustness of the compressor/decompressor against loss in the system (due to noise, erasure and failures in network, hardware or software)? That is, if a small part of the compressed data is lost then the whole chunk of data cannot be recovered because of the different scanning pattern introduced in the paper. This problem is not addressed in this paper. This raises question about the so called ‘better compression ratio’ of the Rasterzip compression method if the increase of the number of copies of the data is required to protect against failures. Note that, in comparing source coding methods, it is crucial to consider the amount of the protection that it is added to the system after compressing the data (e.g., for transmitting the compressed data later). For instance, if we reduce the amount of data by 1/4, but somewhere else we need to keep 2 copies of data what is the point?

The data size shown in Table 1 is quite low, which brings into questions the real compression ratio & comparison in Section 4. Again, no evaluation of the robustness of RasterZip. More thorough evaluation (using different data sets with different sizes) and performance comparison with other algorithms is needed.

What is the cost for applying this method to the existing sys-

tems? In fact, how compatible this method is with current systems? This is an important question because in high speed applications compression/decompression algorithms are typically implemented in hardware.

The effect of using different packet sampling methods on the performance of compression/decompression has not been addressed. This is important because packet sampling methods can affect the number of V-Blocks, and consequently, the efficiency of RLE.

Overall, the above issues can significantly limit the advantage and general applicability of Raster Zip for realistic usage case in existing networks.

Currently it’s unclear how this method is applicable in more general cases. You may want to compute the entropy of the sources of data (network traffic measurement data) to provide a lower bound on the achievable compression ratio and to evaluate the statistical correlation among symbols of the source. The latter could lead to using other loss less and variable length coding compression algorithms in this application.

Reviewer #2

Summary: The paper presents the design of a new compression scheme targeted at network flow records. The approach takes the inherent structure of the fields, e.g., IP addresses, temporal locality, e.g., ports accessed, and spatial locality, e.g., few protocols. The idea is to group records that match on fields, then layout data in columnar format, and compress using a modified form of run-length encoding. This results in better compression compared to conventional schemes, while also ensuring fast decompression to ensure targeted queries can be answered in real time over compressed data.

Strengths: Interesting problem. Interesting design. Simultaneously meeting high compression and responsiveness for answering queries over compressed data is hard!

Weaknesses:

- Approach relies on key innovations in the authors’ prior work. This in itself is not such a problem, except that the paper does not describe the prior work in any great detail so it really hard to understand to what extent the overall design depends on inherent advantages, or disadvantages, of prior work.
- Description is very vague in part. Several design choices and parameters are not explained.
- For all the sophistication in design, benefits appear to be small.

Comments to authors: Section 2: You need to start by *precisely* describing what the goals for your design are. Then you talk about prior approaches and what they lack.

The third limitation, "they are not able to exploit indexes to retrieve attribute values stored at a specific position from a compressed block" makes no sense without appropriate context on what you expect your system to offer.

Why don't you show decompression speed in Figure 2? Are you assuming there is a functions of the other two axes (compression ratio and speed)?

Section 3: What is oLSH? A detailed description is *very* crucial to understand what kind of index it compiles, how the index is used to retrieve appropriate data to de-compress etc.

The entire description of the algorithm uses the example of a simple field (source) IP address). It is not at all clear from the description how this extends to the multiple dimensions in a record (i.e., the other fields). Also, the exact details on a collection of multi-dimensional records are compressed to are not clear. In particular, are you compressing attribute by attribute? Or are you doing multiple attributes at once?

How did you choose 32 as the number of runs to try and encode together? What are the tradeoffs imposed by this choice?

Why do you only focus on run lengths ≤ 2 ? Why this choice? Why not also include run lengths of 2 in your approach? Does this lower compression ratio, or some such? If so, mention clearly.

You say "block span" is the number of bytes of uncompressed data stored per block. Yet, the formula you show soon after does not seem to compute that; the second part of the summation on the right hand side encode *compressed* bytes. Am I missing something?

3.2: Can you give an example of a query? And drive the rest of the discussion based on this example?

This section was made unnecessarily hard to read because of lack on sufficient description of the design of the index. I was left wondering how you are able to derive the rows of interest? Would you be able to do this for arbitrary queries, e.g., multi-attribute range queries?

As before I did not understand how decompression would work in a multi-attribute scenario. The description is based on a single attribute.

You describe an efficient partial decompression algorithm. But I don't understand why you need it. What constraints lead you to explore this design? Also, given that in Figure 11, partial decompression almost never gets employed does the improved algorithm really help?

Section 4: You chose a block size of 4000 records. Why? Any tradeoffs imposed by this choice?

Improvements from Rasterzip seem small (E.g. table 2).

How do you pick the training and test sets in Section 4.3? Do you do n-fold cross validation? How important is it to get an accurate model during the profiling stage?

Figure 14 seems disappointing. The adaption decompression doesn't seem to help at all. It appears from the picture that always picking full decompression is as good (or bad). Can you then justify why you need an adaptive scheme?

You say that Rasterzip offers lower response time than LZ0 for a

small number of "rare" queries. But the example you give, queries for port 80 traffic, seems hardly rare!

On the whole, this is an interesting problem. But the description is too muddled and several key aspects are not well justified. The evaluation does not seem to convince me that the approach is worthwhile.

Reviewer #3

Summary: This paper describes a new compression technique to compress network traces. The authors have two goals for the compression technique: 1. it needs to allow real-time compression as the network flows pass through a router, and 2. it needs to support fast retrieval by allowing decompression of partial blocks.

Strengths: The goal of RasterZip is to design an archival/compression technique that works with real time traffic, that takes into account the special properties of the data, and makes it easy to decompress.

The paper builds on the authors previous work called oLSH that permutes incoming flow records to place similar records together. The authors tweak oLSH to place similar IP addresses together, to leverage the common prefix structure. They then design an archival technique that stores records according to column matrix instead of the row matrix (in other words, the first symbol of all IP addresses in the matrix are stored as a block, then the second symbol, etc).

This is a neat optimization trick: Since most IP addresses have common prefixes, this column based parsing + using run length encoding (where the repeated symbols are coupled together) provides some savings in storage

Decompression is performed either by fully decompressing the blocks or only partially decompressing the blocks. The trick to partial decompression is to maintain a subblock range, and use this range to decompress only certain blocks. The authors design an adaptive decompression strategy that decides on the fly to either use the partial decompression or full decompression.

The authors evaluate this technique to show that the speed of archival is as good as the previous technique, with a high speed of retrieval

Weaknesses: First the topic is a bit tangential. Second, I have some concerns about the motivation. The authors claim to use specific data patterns of network flow data to improve archiving. But the only feature they use is the fact that IP addresses have common prefixes. Can you do something simpler; for example, hash all 192. to a single value, etc?

The authors also claim that the archival needs to be done at real time. I don't understand why. Retrieval can only be done after the data has been indexed (since that is the model used; the query response is retrieved from the index and the corresponding row is fetched from the archives). Given this model, why can't the data be written to disk and then archived. I don't know that archival is easier when the data is stored vs being realtime, but the authors make a big deal about this and I am not sure what the motivation for this is.

Comments to authors: This paper needs a problem statement. What is the problem in current network flow archival systems? What is the retrieval latency of fetching data from these systems?

Is it poor? Does it even exist? Instead, the paper dives into the solution

Because of the lack of a problem motivation, I wasn't able to imagine what the queries would look like. What are some of the common queries issued to these archival records?

You evaluate the speed of retrieval for retrieving records that contain certain port numbers. Are these the most common type of queries? Is the performance tied to the query itself. Is it likely that for a different set of query, the retrieval performance will be poorer? Or does the retrieval speed only depend on the number of records fetched?

Similarly, the evaluation of the adaptive decompression strategy is performed by retrieving all records that contain a set of IP address. Again, how common is this query? Surely, the performance of adaptive decompression strategy will depend on the queries? Will the strategy work just as well if other queries are used?

Why is an adaptive decompression strategy even needed. Why not do both in parallel; i.e., perform full decompression, but give priorities to the sub-blocks that likely contain the record.

Reviewer #4

Summary: This paper presents RasterZip, a compression algorithm specifically tailored to flow records (i.e., Netflow).

Compression begins by reordering arriving flow records by the first column value, to maximize run lengths with similar prefixes in the column direction. The compression algorithm progresses through the resulting matrix column-by-column, using a clever run-length encoding variant that produces a series of blocks of two possible types: V-Blocks (verbatim) for subsequences that don't RLE-compress well, and B-Blocks (bitmaps) for subsequences that do.

The block design provides low memory footprint, fast decompression of uncompressable ranges, fast scanning of blocks due to quick extraction of block lengths, and fast computation of a block's span, i.e. the number of runs it contains.

For decompression, RasterZip supports two approaches: full and partial decompression. In the former all blocks get decompressed, in the latter only those blocks relevant to the rows resulting from the user's query. The authors present a way to quickly identify which blocks need decompression given the need to access a given row, using a bitmasking approach that approximates row locations, mapping those locations in a grid-like fashion.

Next is an adaptive algorithm for automatically deciding whether to decompress fully or partially, based on query selectivity and the compression rate of the required blocks (in good compression, full decompression may be better).

For evaluation, the authors take a six-day NetFlow trace from a hosting environment (232M records) and a two-month NetFlow trace from a production network (1.2B records). They compare compression ratio, insertion rate, adaptive decompression performance (compared to full and partial), and overall query performance, finding that RasterZip at least on par with LZO or clearly beating it.

Strengths: The paper is well-executed and thorough, addressing a relevant problem.

Weaknesses: Given the comparison to LZO throughout the paper, it would help to get a bit of context on how it works. The same applies to oLSH as well as your index construction. You refer to the relevant papers, but getting at least a quick summary (perhaps instead of the algorithm sketches) would help. The figures need a bit of work, both in terms of clarity and legibility.

Comments to authors: Nice paper, I enjoyed the read and found it mostly smooth but got stuck a few times, largely when trying to digest the diagrams. I would love to know how well this algorithm can perform on pcap traces. Also, do you plan to release the code?

The example in Figure 1 would be more effective if you gave some intuition regarding how LZO leverages data homogeneity. It's intuitively clear that sorting by increasing destination port number increases homogeneity, but (without prior knowledge) not how compression algorithms benefit during dictionary construction etc. More generally, as mentioned above, a broader recap on LZO would help.

The following seems to be a bug: in the description of V-Blocks you say only lengths of at least three are stored explicitly, but Figure 6 says lengths greater than 1.

In Figures 7 and 9 it would be very helpful if you could work in the difference between subblocks and runs (i.e., show somehow extra-clearly that 1-13 doesn't refer to runs but subblocks, and ideally how the IP address octets you're showing on the right correspond to the stored subblocks). This confused me at first.

By the time we get to Section 4 (Evaluation), it's unclear what exactly is a query, how you compute row indices, and what is the nature of indexes.

It's also unclear to me how you generalize from a single-attribute column (e.g. an IP address) to a full multi-attribute flow record. Several questions come to mind here, for example: How do you pick the sorting order? Widest-to-narrowest column? How does the selection of this sort order balance with index size? Does the increase in fragmentation from left to right in the columns mean that the approach only works for relatively low-dimensional data sets? If so, where's the sweet spot?

Minor: odd that on your test machine /etc/services contains only 311 ports – mine shows over 10K.

Reviewer #5

Summary: The paper proposes Rasterzip, a compression/decompression technique custom build for network data. Interesting ideas and good results. Paper could have been written better.

Strengths: Interesting idea Good evaluation and results

Weaknesses: Paper could be written better

Comments to authors: Overall the ideas were interesting and the results show good performance improvement. The paper could have been written better in some parts. Details below.

- The data sorting methodology was not clear. While a reference has been provided, it would be useful to explain the core of the idea in the paper itself. I did not follow the explanation of

figure 1. More details are needed. How do you maintain timeliness and answer timeliness queries if you reorder the data? Some explanation is required. The sorting is based on which field, if you sort based on one field, some other fields may not yield enough compression. So, how do you decide which field to sort on?

- What was the reason for choosing V blocks with runlengths upto 2. Why not 3 or even 1?
- I could not also clearly follow section 3.2 including the figures, especially implementing the partial decompression. More effort is needed to explain the process better. It may be useful to take a concrete example (involving Ip addresses or ports) to explain all this.
- In section 4.2, What is the 8x8 bit map? It was not clear.
- Figure 5: Input stream has 15, while figure has 16
- Figure 14, use different color/line type, its not easy to distinguish between full and adaptive. It appears that full decompression is what is being done in majority of the cases of figure 14. So, in reality when using the traces, did the adaptive strategy really help in terms of response times etc?

Response from the Authors

We made a number of changes to improve the readability of our paper based on the comments of the reviewers. In particular, we made the following main changes: 1) in the introduction, we explained

better the motivation and contributions of our work; 2) in Section 2, we made more clear how bitmap indexes work and extended the description of oLSH; and 3) we changed the organization of the paper, which originally used one big section explaining both compression and decompression, splitting this content into two sections and several subsections.

In addition, to address the comment about the choice of the parameter of the number of runs, we introduced a new paragraph (Additional properties of the encoding scheme) at the end of Section 3.2 explaining why the number of runs is equal to 32. To address the question about the resiliency of our scheme to errors, we introduced a new section (Section 4.4, Resiliency to errors) where we explain that errors cannot be detected by RasterZIP directly, but in a higher layer.

Besides, we changed Section 5.4 to address the comment of the reviewers about how the evaluation results would differ if multi-attribute queries were used instead of single-attribute queries. The short answer is that using single- or multi-attribute queries does not play a role in the comparison between LZO and RasterZIP because it would only affect the index lookup time equally for both compressors. In the same section, we also explained better why queries on port 80 in the data from the hosting environment represent a special case.

Finally, in Section 5 we introduced additional evaluation results in which we show that RasterZIP produces more compact archives than bzip2 too. This was a positive finding we had not explored before the submission. Based on this, we also updated Figure 2 in Section 2.