Lossless Compression of High-Frequency Voltage and Current Data in Smart Grids

Andreas Unterweger and Dominik Engel
Salzburg University of Applied Sciences,
Josef Ressel Center for User-Centric Smart Grid Privacy, Security and Control
Urstein Süd 1, 5412 Puch/Salzburg, Austria
Email: andreas.unterweger@en-trust.at

***Abstract*—In smart grids, both, low-frequency and high-frequency measurements are performed in households for a variety of use cases. While several compressibility studies of this data have been conducted in the literature, lossless compression of high-frequency data has not yet been covered. In this paper, high-frequency voltage and current data is processed with a selection of low-complexity compression algorithms to find that the data are not equally compressible. Further, it is found that the compression performance varies with resolution as well as between households and data sets. Nonetheless, the use of compression is practically viable for the current channels of the evaluated data sets at 16 and 50 kHz, respectively.**

***Keywords*-Smart Grid; Compression; High Frequency; Voltage; Current**

I. Introduction

Intelligent energy networks, or *smart grids*, are envisioned to modernize energy systems worldwide, on the one hand by enabling communication between the involved entities in the grid, on the other hand by collecting fine-grained sensor measurements throughout the grid. In smart grids, the data collection also applies to the low-voltage part of the grid, which makes smart meters and other measurement units important components of the smart grid [1]–[3].

The amount of data produced by smart meters is considerable [3], especially due to the vast numbers of meters that are in the field. At this point in time, the resolution produced by standard smart meters is rather coarse grained with 15 minute in many countries. There are other measurement units that will produce higher frequency data, which will be measured in the range of kilohertz, e.g., *phasor measurement units* [4]. Due to the development of information technology, smart meters will likely be capable of higher-frequency measurements in the foreseeable future.

As an example for the amount of data accrued in the grid for such measurements, consider the collection of data for voltage and current, with 16 bit precision per value at a frequency of 10 kHz. This will amount to 40 kB/s, i.e., approx. 39 KiB/s per meter. Transmitting this kind of data will require considerable bandwidth for the associated communication links, especially at the receiving, i.e., collecting end. Furthermore, storing the data in an uncompressed format for 100 million meters would result in approx. 4.559 EiB per year ($10^8 \cdot 40.000$ bytes/s $\cdot\, 365$ d/a $\cdot\, 3600$ s/d ≈ 4.559 EiB/a). Although this is only an illustrative example, the case for compression as a means to address the transmission and storage challenges that are typical for a big data application [3] is clear.

It comes without surprise that there are thus numerous suggestions for compressing data in energy metering. In the area of high-frequency compression, there is an exclusive focus on lossy compression (e.g., [4]–[7]), which means that in the decompression process, the data cannot be retrieved exactly. Depending on the extent of data loss, this of course may have detrimental effects on some use cases that require measurement data to be as exact as possible.

Lossless compression can remedy this problem, as it only aims at utilizing redundancies in the data set to maximize the entropy of the compressed representation, i.e., after decompression the exact same data is retrieved. Lossless compression is mainly proposed for low-frequency data, e.g., [8], [9], and, to the best of the authors' knowledge, has not been studied in any meaningful way for high-frequency data sets.

In this paper, the utility of lossless compression algorithms for high-frequency metering data is investigated. Lossless compression algorithms allow universal use of the metering data at a later stage, but still make transmission and storage feasible in the real world. As meters used in the field are typically devices of low computing power, suitable compression approaches need to be economic in terms of their computational and memory demands. The proposed methods are applied to publicly available high-frequency data sets to allow for reproducible results.

The main contributions of this paper are (i) the assessment of **lossless** compression of high-frequency smart meter data; (ii) a viability analysis regarding the compressibility of high-frequency voltage and current data as opposed to consumption values in Watts or Watt-hours; and (iii) recommendations on the compression of high-frequency smart meter data in an environment that is contrained in terms of both, computational power and transmission bandwidth.

This paper is structured as follows: In Section II, lossless compression approaches which are designed for operation on smart meters are presented. In Section, III, the high-frequency data sets used for the evaluation in Section IV are described.

II. Compression Approaches

In this section, relevant lossless compression approaches are presented. The selection of algorithms is based on the constraints which are typical for smart meters, i.e., low computational power and limited memory. From the available light-weight approaches, three of those designed explicitly for the use on smart meters are selected and described below – A-XDR, LZMH and DEGA coding. This selection is based on related work [10].

A. A-XDR Coding

A-XDR [11] stores values with a pre-defined length in binary. Each value uses the same number of bits. Fractional values, have to be converted to integers before storage [9], [10]:

$$v' = v10^d, \tag{1}$$

where d is the number of decimal places after the decimal point, v is the fractional value and v' is the (converted) integer value that can be encoded by A-XDR.

With a length of 16 bits per value, the ranges $[0; 65535]$ or $[-32768; 32767]$ can be represented, depending on whether only positive or positive and negative numbers may occur in the input. For values with two decimal places, the ranges change to $[0; 655.35]$ and $[-327.68; 327.67]$, respectively.

While A-XDR is not a compression algorithm per se, it is one of the simplest possible representations for current and voltage values and thus serves as a reference point. Furthermore, it is both, easy to implement and fast to process, making it suitable for smart meters.

B. LZMH Coding

LZMH coding [8] processes streams of input values character by character and stores a small history of previously processed characters in a buffer. Re-occurring groups of characters are encoded as references to this buffer by specifying the position and length of the group. The remaining data is encoded based on the probability of each processed character – if it occurred often in the input stream so far, it gets assigned a short representation, otherwise it gets a longer one.

Since LZMH coding processes the input character by character, no additional conversions are necessary. The group matching as well as the probability estimation are trivial based on the character count from the input, allowing for low computational complexity. Since the size of the history buffer is very limited, the memory requirements are also very low, making LZMH coding suitable for smart meters.

C. DEGA Coding

DEGA coding [9] calculates the differences between two consecutive values under the assumption that they are small. If it holds, a short representation is used, while, for large differences, a longer representation is required. An additional entropy coding stage further compresses consecutive representations of very small values.

Like A-XDR, DEGA coding is designed for integer input values, requiring the application of Equation 1 before processing. The complexity of DEGA coding is analyzed in detail in [9] – it is designed for use on smart meters.

The efficacy of DEGA coding relies on small differences of consecutive input values. The original publication [9] based this core assumption of the algorithm design on an analysis of the TUD tracebase [12] and MIT REDD [13] low-frequency data sets. Since this paper uses high-frequency data as described in detail in the next section, it is not clear whether the DEGA assumption still holds. Thus, a discussion of this aspect in the evaluation in Section IV is required.

III. High-Frequency Data

In this section, two data sets containing high-frequency voltage and current data of households are described, both of which are subsets of the MIT REDD and UK-DALE data sets, respectively, and dedicated to evaluation purposes. Furthermore, it is described how the data from these data sets is pre-processed so that it can be input directly into the compression algorithms described in Section II. The detailed commands used for pre-processing are listed in Appendix A for reproducability.

A. MIT REDD High-Frequency Raw Data

The MIT REDD data set [13] contains a subset of high-frequency raw data of two houses. For each house, one voltage and two current channels from the mains are available. Each channel contains approximately one-and-a-half hours worth of raw A/D converter data at a sampling rate of 50 kHz.

The actual ADC values are grouped into 21 files per channel per house. They are losslessly compressed with *bzip2* (http://bzip.org), i.e., for processing, they have to be decompressed. Each decompressed file is a sequence of 32-bit floating-point values in little endian byte order, with interspersed time stamps. The latter start with four indicator bytes and contain 8-byte time stamps, i.e., they are 12 bytes in size each and removed before further processing.

Since the majority of compression approaches described in Section II have been designed for compressing primarily consumption values in W or Wh, they have difficulties processing negative values. The latter occur frequently in the high-frequency wave form data.

In order to avoid issues involving negative values, for each file, during processing, the smallest value, i.e., the negative number with the largest absolute value, is added to each value of that file. This makes all values positive and therefore unproblematic for the compression algorithms. Furthermore, it does not alter to compressibility of the values since their entropy does not change. Thus, this change does not impact the results and conclusions of the evaluation in Section IV, but enables them in the first place.

B. UK-DALE High-Frequency One-Week Data

The UK-DALE data set [14] contains a subset of high-frequency (diaggregated) raw data of one house (DOI 10.5286/UKERC.EDC.000002). For this house, one voltage and one current channel are available. Each channel contains one week of raw A/D converter data at a sampling rate of 16 kHz.

The actual ADC values are grouped into 169 files, containing both, voltage and current data. The files are losslessly compressed with *FLAC* (https://xiph.org/flac/), i.e., for processing, they have to be decompressed and split into the two individual channels. Both, decompressing and splitting, are performed using *avconv* (https://libav.org/avconv.html) from *Libav* (https://libav.org/). Each decompressed channel (as a file) is a sequence of 64-bit floating-point values between -1 and 1.

In order to obtain the actual voltage and current readings from these floating-point values, each value $\tilde{v}_{i,j}$ of a decompressed file i is scaled:

$$v_{i,j} = 2^{31} \Delta step \cdot \tilde{v}_{i,j}, \tag{2}$$

where 2^{31} is the number of ADC steps per sign (plus and minus, yielding a total of 2^{32} ADC steps) and $\Delta step$ is the ADC step size, which is $1.90101491444 \cdot 10^{-7} \frac{\text{V}}{\text{step}}$ for the voltage channel and $4.9224284384 \cdot 10^{-8} \frac{\text{A}}{\text{step}}$ for the current channel, respectively. To avoid issues with negative values, an offset is added to the scaled values $v_{i,j}$ as described for the MIT REDD data set above.

IV. Evaluation

In this section, the evaluation methodology for the compression approaches presented in Section II is explained. Subsequently, the compression results achieved for the test data described in Section III are presented. Results on run time are briefly discussed in Appendix B.

A. Methodology

To evaluate the compression performance of the high-frequency data, it is processed in three steps as follows:

1) **Separation**: From the data, the individual channels are extracted on a file-by-file basis as described in Section III. This ensures that chunks of approximately equal size for each channel are available separately for pre-processing.
2) **Pre-processing**: The raw channel data consists of floating-point values which are unsuitable for the compression algorithms – they are designed to process input values with a constant number of decimal places. Thus, as a pre-processing step, all values are converted into a decimal representation with two decimal places after the decimal point. This process is, in fact, lossy and can be adapted based on the measurement precision and/or use case. For this evaluation, 16-bit precision of the A/D converters is assumed, allowing for values of approximately ± 328 with two decimal places after the decimal point (see Section II).
3) **CSV formatting**: The pre-processed values with two decimal places are stored as CSV (comma-separated values) files. These files are used as the input for all compression algorithms so that their compression performance is comparable. Each CSV file represents the data of one channel of one file from one data set.

Each pre-processed file i with size s_i is compressed with all compression algorithms a, resulting in compressed files of sizes $\hat{s}_i^a$. From the compressed sizes, the compression ratios r_i^a are calculated as

$$r_i^a = \frac{\hat{s}_i^a}{s_i} \tag{3}$$

To evaluate the compression performance p^a of all files of a channel for any compression algorithm a, the compression ratios are combined as follows. As opposed to [9], which use the arithmetic mean of compression ratios, for this evaluation, the harmonic mean is used:

$$p^a = \frac{n}{\sum\limits_i \frac{1}{r_i^a}}, \tag{4}$$

where n is the number of input files i. This allows for a more realistic assessment of the overall compression performance – input files of approximately equal size, but different compressibility can be compressed by a common ratio (the harmonic mean, p_a) when combined.

In order to evaluate the compression performance at lower frequencies, e.g., a frequency that is m times smaller, the input files are processed as described above, but, additionally, m consecutive values $v_{i,j}$ in each file i are combined using the arithmetic mean

$$v'_{i,j'} = \frac{\sum\limits_{j=j'm}^{(j'+1)m-1} v_{i,j}}{m}, \tag{5}$$

resulting in m' lower-frequency values $v'_{i,j'}$. These values represent an approximate lower-frequency sampling of the high-frequency values. To evaluate the lower-frequency sampling, $v'_{i,j'}$ are stored as CSV and compressed instead of $v_{i,j}$.

For all evaluations of the compression approaches, the implementation from [10] is used. All measurements are carried out on a 64-bit x86 virtual machine running Ubuntu 14.04 on a Linux kernel with version 3.13. All pre-processing described in Section III and above is carried out with the latest GNU *Coreutils* (http://www.gnu.org/software/coreutils/coreutils.html) available over *apt* as of March 2016.

B. Results

The compression results for the MIT REDD and UK-DALE data sets are presented separately. Figure 1 depicts the median compression ratios for the MIT REDD high-frequency raw data set channels at different frequencies. Minimum and maximum compression ratios are indicated by horizontal bars. Three main observations can be made.

First, DEGA coding (Unterweger15a [9], black) outperforms the other approaches in terms of compression performance. This is true for the voltage channel (top, filled triangles) and both current channels (middle, filled rectangles and bottom, filled circles) as well as for all resolutions but 100 Hz. The relative performance of LZMH coding (Ringwelski12a [8], dark grey) and A-XDR (light grey) is comparable to what has been found in prior work [10].

Second, there is a performance difference between the two houses, 3 (left) and 5 (right). While the voltage channels (top, filled triangles) are practically identical in terms of compression ratio, the two current channels (middle, filled rectangles and bottom, filled circles) are different. Although the shape of the curves is similar, the absolute values differ. Due to the logarithmic scale, the differences appear smaller than they are. However, the differences between the houses are relatively small, i.e., the compression performance of the respective channels is similar.

Third, the performance of all compression algorithms – DEGA coding (Unterweger15a [9], black) and LZMH coding (Ringwelski12a [8], dark grey) – depends on the resolution in a similar way: For very high frequencies (50 kHz), compression performance is good, i.e., around a compression ratio of 10 for the current channels, and decreases with frequency until around 500 Hz. This shows that the data contains little to no noise, i.e., it is compressible at its original resolution, and that the decrease in compression performance is due to an decrease of temporal correlation at lower resolutions.

At a frequency of 100 Hz, the compression ratio increases again, but much more significantly for LZMH coding (Ringwelski12a [8], dark grey). Since 100 Hz is just below the required sampling frequency of a 50 Hz signal [15], the inherent band limitation decreases the signal entropy in the time domain, thus making it more compressible. This effect becomes stronger for frequencies below 100 Hz (not depicted). Note that all results at frequencies of 100 Hz and below are irrelevant in practice since subsampling renders most of the data useless. However, these frequencies are useful when the data at hand is accumulated instead of sampled, as analyzed in [8]–[10].

The results for the second data set, the UK-DALE high-frequency one-week data set, are depicted in Figure 2. The compression performance depends on the resolution (frequency) of the input data in relatively the same way as for the MIT REDD high-frequency raw data set, with the exception of low frequencies, which are not practically relevant, as explained above. Again, DEGA coding (Unterweger15a, black) outperforms the other approaches. Two additional observations can be made.

First, there is a significant difference between the voltage (left) and current channels (right) – the voltage channel is less compressible. This can also be observed for the MIT REDD high-frequency raw data set (see Figure 1), albeit to a smaller extent. This result, which is due to the lower entropy of the current channel, may either be explained by voltage spikes due to inductive loads or by the influence of other households on the same transmission line – changes in these other households leads to an addition of noise to the voltage channel, thereby increasing the entropy of each household, while decreasing the entropy overall, i.e., for all households on a transmission line. A detailed analysis of the causes, however, remains future work.

Second, comparing channels of the same type (e.g., voltage only) between the UK-DALE high-frequency one-week data set (Figure 2) and the MIT REDD high-frequency raw data set (Figure 1) reveals about the same level of change as is observed for channels of the same type between different houses of the MIT REDD high-frequency raw data set. This allows for the conclusion that the data-dependent compression ratio variability is non-negligible.

V. Conclusion

The compressibility of high-frequency voltage and current measurements has been studied for a number of smart-meter-specific compression algorithms. Three main results can be highlighted. First, current data can be compressed significantly better than voltage data. Second, the results depend on the resolution of the data, where a higher frequency yields a better compression ratio (90% data reduction for voltage data) – excluding any resolutions below the practically relevant sampling limit. Third, the compression algorithm proposed by Unterweger and Engel [9] outperforms the others for high-frequency data. In summary, compressing data from current channels with this approach is feasible at high frequencies, while compressing data from voltage channels is likely not worth the effort due to the small compression ratio given the required computation time.

Acknowledgment

The financial support by the Austrian Federal Ministry of Science, Research and Economy and the Austrian National Foundation for Research, Technology and Development is gratefully acknowledged.

References

[1] Z. Fan, P. Kulkarni, S. Gormus, C. Efthymiou, G. Kalogridis, M. Sooriyabandara, Z. Zhu, S. Lambotharan, and W. H. Chin, "Smart grid communications: Overview of research challenges, solutions, and standardization activities," *IEEE*

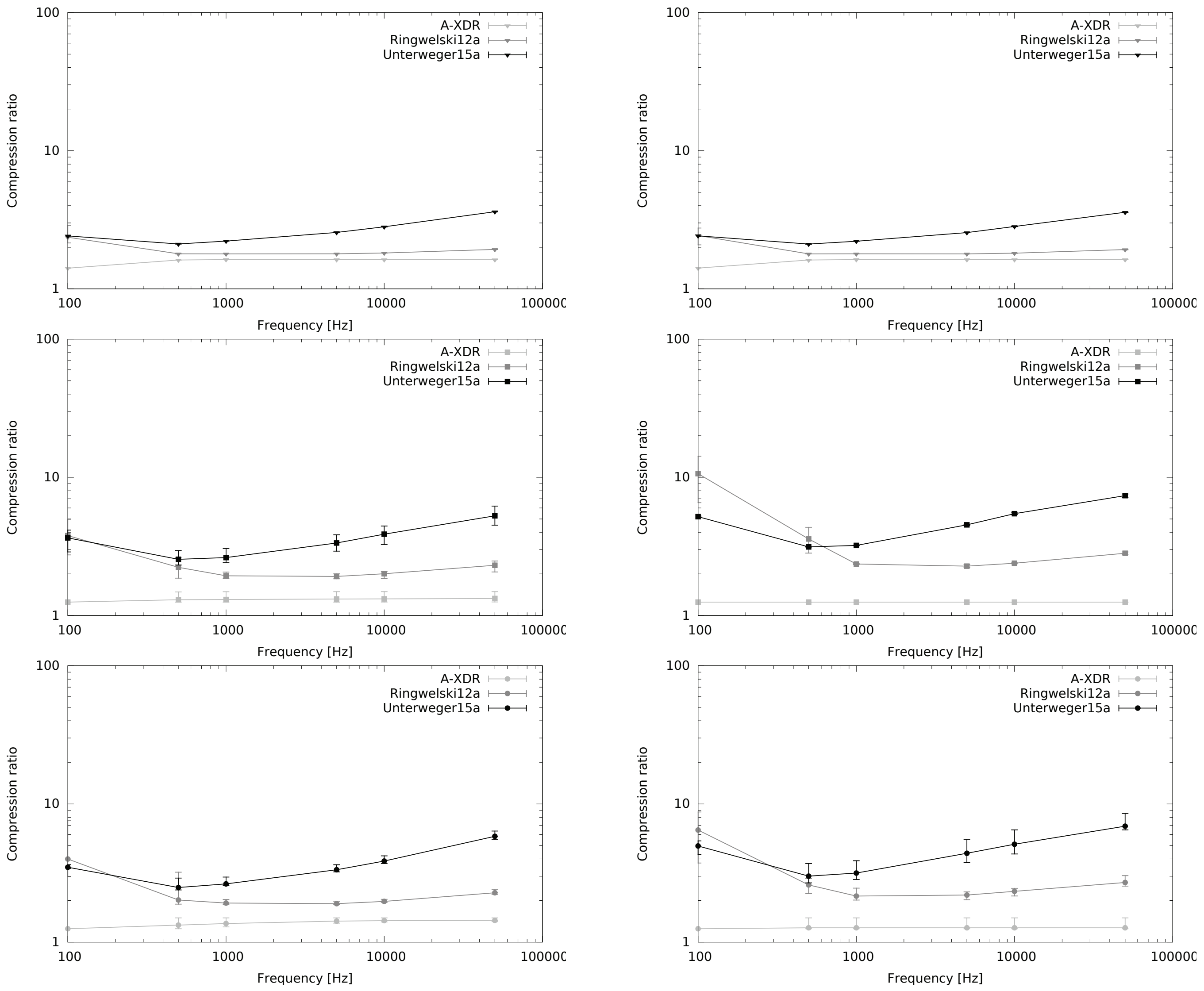

Figure 1. Compression performance of different algorithms for houses 3 (left) and 5 (right) from the MIT REDD high-frequency raw data set: The results for voltage channels (top, filled triangles) differ from those for current channels 1 (middle, filled rectangles) and 2 (bottom, filled circles). The results also differ between the two houses, but not for all channels.

Communications Surveys and Tutorials, vol. 15, no. 1, pp. 21–38, 2013.

[2] X. Fang, S. Misra, G. Xue, and D. Yang, "Smart grid - The new and improved power grid: A survey," *IEEE Communications Surveys and Tutorials*, vol. 14, no. 4, pp. 944–980, 2012.

[3] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mobile Networks and Applications*, vol. 19, no. 2, pp. 171–209, 2014.

[4] P. H. Gadde, M. Biswal, S. Brahma, and H. Cao, "Efficient Compression of PMU Data in WAMS," *IEEE Transactions on Smart Grid*, 2016, to appear.

[5] X. Tong, C. Kang, and Q. Xia, "Smart Metering Load Data Compression Based on Load Feature Identification," *IEEE Transactions on Smart Grid*, 2016, to appear.

[6] J. C. S. de Souza, T. M. L. Assis, and B. C. Pal, "Data compression in smart distribution systems via singular value decomposition," *IEEE Transactions on Smart Grid*, 2015, to appear.

[7] J. Cormane and F. A. d. O. Nascimento, "Spectral shape estimation in data compression for smart grid monitoring," *IEEE Transactions on Smart Grid*, vol. 7, no. 3, pp. 1214–1221, 2015.

[8] M. Ringwelski, C. Renner, A. Reinhardt, A. Weigel, and V. Turau, "The Hitchhiker's guide to choosing the compression algorithm for your smart meter data," in *2012 IEEE Inter-*

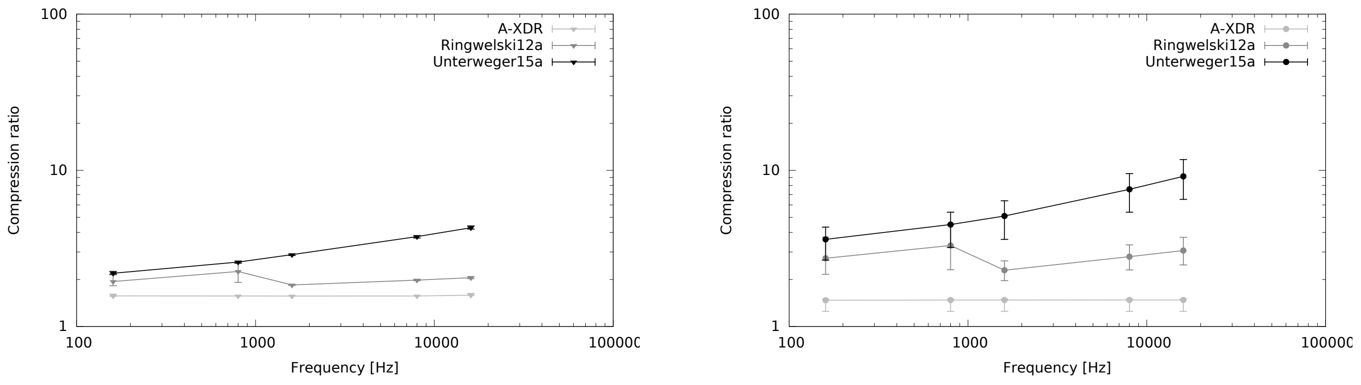

Figure 2. Compression performance of different algorithms for the *volts* (left) and *amps* channels (right) from the UK-DALE high-frequency one-week data set: The results between channels differs, but the resolution dependency is similar to that observed for the MIT REDD high-frequency raw data set in Figure 1.

national Energy Conference and Exhibition (ENERGYCON), sep 2012, pp. 935–940.

[9] A. Unterweger and D. Engel, "Resumable Load Data Compression in Smart Grids," *IEEE Transactions on Smart Grid*, vol. 6, no. 2, pp. 919–929, 2015. [Online]. Available: http://dx.doi.org/10.1109/TSG.2014.2364686

[10] A. Unterweger, D. Engel, and M. Ringwelski, "The Effect of Data Granularity on Load Data Compression," in *Energy Informatics - 4th D-A-CH Conference, EI 2015, Karlsruhe, Germany, November 12-13, 2015, Proceedings*, ser. Lecture Notes in Computer Science, S. Gottwalt, L. König, and H. Schmeck, Eds., vol. 9424. Springer, 2015, pp. 69–80. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-25876-8{_}7

[11] "Distribution Automation Using Distribution Line Carrier Systems – Part 6: A-XDR Encoding Rule," 2000.

[12] A. Reinhardt, P. Baumann, D. Burgstahler, M. Hollick, H. Chonov, M. Werner, and R. Steinmetz, "On the Accuracy of Appliance Identification Based on Distributed Load Metering Data," in *Proceedings of the 2nd IFIP Conference on Sustainable Internet and ICT for Sustainability (SustainIT)*, 2012, pp. 1–9.

[13] J. Kolter and M. J. Johnson, "Redd: A Public Data Set for Energy Disaggregation Research," in *Workshop on Data Mining Applications in Sustainability (SIGKDD)*, aug 2011, pp. 1–6. [Online]. Available: http://redd.csail.mit.edu/kolter-kddsust11.pdf

[14] J. Kelly and W. Knottenbelt, "The UK-DALE dataset, domestic appliance-level electricity demand and whole-house demand from five UK homes," *Scientific Data*, vol. 2, p. 150007, mar 2015. [Online]. Available: http://dx.doi.org/10.1038/sdata.2015.710.1038/sdata.2015.7

[15] A. V. Oppenheim, *Discrete-Time Signal Processing*, ser. Pearson Education Signal Processing Series. Pearson Education, 1999.

APPENDIX A. PRE-PROCESSING COMMANDS

This appendix describes the exact algorithms used to pre-process the data of the MIT REDD high frequency raw and UK-DALE high-frequency one-week data sets. The common commands to offset negative values are listed separately. All code is provided in the form of GNU *Bash* (https://www.gnu.org/software/bash/) commands using the software mentioned in Sections III and IV.

A. MIT REDD High-Frequency Raw Data

The code in Listing 1 converts one file of one channel of one house, e.g., `house_3/current_1/1303091049.bz2`, the path of which is stored in the variable `inputfile`, into a CSV file, the path of which is specified in the variable `outputfile`.

```
1 bzip2 -dkc "$inputfile" | \
2   hexdump -ve '1/1 "%.2X"' | \
3   sed 's/74696D65................//g' | \
4   xxd -r -p | \
5   od -f -An -w4 -v | \
6   sed 's/,/\./g' | sed 's/ //g' | \
7   awk '{ printf("%.2f\n", $1); }' | \
8   sed 's/-0\.00/0\.00/g' > \
9   "$outputfile" 2>/dev/null
```

Listing 1. Pre-processing commands to convert an input file of the MIT REDD high-frequency raw data set to a CSV output file with two decimal places for compression.

In short, the input file is decompressed (line 1), time stamps are removed (lines 2-4), the data values are formatted as decimal numbers in ASCII (line 5) and this output is cleaned so that it resembles the content of a valid CSV file (lines 6-8). This content is stored in the output file (line 9). A more detailed explanation follows.

In line 1, the input file is decompressed (`d` parameter) to *stdout* (`c` parameter) with `bzip2`. The input file is not deleted (`k` parameter). The decompressed output, a sequence of floating-point values with interspersed time stamps is piped to `hexdump` in line 2.

`hexdump` formats the individual bytes as two upper-case hexadecimal digits per byte (`e` parameter with a *printf*-like format string). Values are formatted as one single string of digits (*iteration count*/*bytecount* notation) for easier subsequent filtering. Repeating data is displayed instead of producing * characters in the output stream (`v` parameter). This data is passed to `sed` in line 3.

`sed` filters the time stamps. All instances (`g` suffix) of four indicator bytes (`74696D65` hexadecimal), followed by an eight-byte timestamp (16 arbitrary hexadecimal digits) are substituted (`s` prefix) by an empty string, eliminating all time stamps from the data. The remaining data values are piped to `xxd` in line 4.

`xxd` re-converts (`r` parameter) the continuous input (`p` parameter) of hexadecimal characters to floating-point values. After line 4, the content of the pipe is a stream of floating-point values without time stamps. Each value is four bytes in size.

In line 5, the single-precision floating point values (`f` parameter) are formatted as decimal numbers with `od`. No address offset information (`An` parameter) is printed and line breaks are introduced after every four bytes (`w4` parameter), i.e., after each input value. This yields an output stream of formatted decimal values, separated by line breaks. Repeating data is, again, displayed instead of producing * characters in the output stream (`v` parameter). This data is passed to `sed` in line 6.

The first call to `sed` substitutes (`s` prefix) all (`g` postfix) decimal commas by decimal points in case a german locale is used – the subsequent calls require decimal points in order to properly recognize decimal values. Similarly, the second call to `sed` removes all white space. The remaining formatted decimal values are piped to `awk` in line 7.

`awk` prints the first (and only) input value of each line (`$1`) as a decimal number with two decimal places using its `printf` command and the corresponding format specifier. This results in the rounding of all floating-point values to exactly two decimal places. Any two values are separated by a line break, making the output valid CSV file content with one column of values.

In line 8, `sed` substitutes (`s` prefix) all (`g` postfix) occurrences of *-0.00* by *0.00*. This helps verfification after compression, since the two values are indistinguishable. The still format-compliant CSV file content is finally redirected (> operator) to the output file in line 9.

The error output is suppressed, i.e., redirected to `/dev/null`. In case errors occur, the `$?` variable is set to a non-zero value, which can be checked after the execution of Listing 1.

B. UK-DALE High-Frequency One-Week Data Set

Before pre-processing data files from the UK-DALE high-frequency one-week data set, the ADC step sizes for the voltage and current channels need to be extracted from the included `calibration.cfg` file. The extraction is shown in Listing 2. It assumes that the path to the data set is stored in the variable `data_dir`.

```
1 channels=( "volts" "amps" )
2 declare -a conversion_factors
3 for channel in "${channels[@]}"
4 do
5   conversion_factors[$channel]=\
6     `cat "$data_dir/calibration.cfg" | \
7         grep '^'$channel | \
8         grep -o '= .*$' | sed 's/= //'`
9 done
```

Listing 2. Extraction of the ADC step sizes for the voltage and current channels from the UK-DALE high-frequency one-week data set.

In line 1, an array with the two channel names is declared. In line 2, an associative array (`a` parameter) is declared to store the ADC step sizes for each of these channels. In lines 3-9, for every channel name from the array, the ADC step size is extracted from the `calibration.cfg`. A detailed explanation of these lines follows.

In line 5, the ADC step size for one channel on the left side of the equation is stored in the associative array declared in line 2 with the channel name as the key. The actual ADC step size is determined on the right side of the equation by the execution of the command (enclosed in `` ` ` `` characters) between lines 6 and 8.

Using `cat`, the `calibration.cfg` file is piped to `grep` in line 7. The file has an INI-file-like structure with additional spaces around the = signs. There is only one *calibration* section with three variables – *volts_per_adc_step*, *phase_difference* and *amps_per_adc_step*. The first and the last need to be extracted.

In line 7, the correct line is extracted by searching for the channel name directly after the line start (^ character). This line is processed by `grep` in line 8, where only (`o` parameter) the = sign and all following characters (`.*` specifier) are extracted. Finally, `sed` substitutes (`s` prefix) the = sign and any superfluous white space by an empty string, leaving the actual value, i.e., the ADC step size of the channel.

With the information about the ADC step sizes being extracted, pre-processing can be performed on a file-by-file basis. The code in Listing 3 converts one file, e.g., `vi-1407520800_943011.flac`, the path of which is stored in the variable `inputfile`, into two CSV files, the path prefix of which is specified in the variable `outputfile` – for the two channels, *volts* and *amps* are appended to this path, respectively.

```
1  avconv -i "$inputfile" -filter_complex \
2    'channelsplit=channel_layout=2[FL][FR]' \
3    -map '[FL]' -f f64le "tempvolts" \
4    -map '[FR]' -f f64le "tempamps" > \
5    /dev/null 2>&1
6  for channel in "${channels[@]}"
7  do
8    cat "temp$channel" | \
9      od -t f8 -An -w8 -v | \
10     sed 's/,/\./g' | sed 's/ //g' | \
11     awk '{ printf("%.2f\n", $1 * 2^31 * ' \
12       ${conversion_factors[$channel]}'); }' \
13     sed 's/-0.00/0.00/g' > \
14     "${outputfile}_$channel" 2>/dev/null
15     rm -f "temp$channel"
16 done
```

Listing 3. Pre-processing commands to convert an input file of the UK-DALE high-frequency one-week data set to a CSV output file with two decimal places for compression.

In short, the pre-processing consists of two code parts: In lines 1-5, the input file is decompressed and split into the voltage and the current channel. In lines 6-16, a CSV file is created from the values of each channel similar to Listing 1. A more detailed explanation follows.

In line 1, the input file is fed into `avconv` and a filter chain (`filter_complex` parameter) for separating the two channels (`channelsplit` filter) is set up in line 2. The channel layout is specified as 2-channel audio with a front-left (`FL` specifier) and a front-right (`FR` specifier) channel.

Since the voltage data is contained in the left channel and the current data in the right one, the split channels are, in lines 3 and 4, mapped (`map` parameters) separately to two files named `tempvolts` and `tempamps`, respectively. The format (`f` parameter) of the values in these files is specified as 64-bit floating-point with little endian byte order (`f64le` specifier).

The standard (console) and error output (`2>` operator) is combined (`&` operator with `1` specifier for *stdout*) and ignored by redirecting it to `/dev/null` in line 5. The error-handling code is omitted here for the sake of readability. At the end of line 5, there are now two temporary files with 64-bit floating-point values in the range between -1 and 1.

In lines 6-16, each channel is processed separately. The temporary file corresponding to the processed channel is output with `cat` in line 8 and piped to `od` in line 9. Similar to line 5 in Listing 1, this formats each 8-byte floating point value (`t` parameter with `f8` format specifier) as a decimal number and puts a line break after every 8 input bytes (`w8` parameter), i.e., after every data value.

The formatting and cleaning to produce CSV-compliant output in lines 10-14 is identical to lines 6-9 of Listing 1, with the exception of two additional multiplications in lines 11 and 12. Here, the values in the range between -1 and 1 are converted to voltage and current values, respectively, by applying Equation 2. The value of $\Delta step$, which depends on the channel, is read from the associative array created in Listing 2 with the channel name as key.

Finally, in line 15, the temporary file with intermediate values for the channel is removed using `rm`. After line 16, there are two CSV files – one with the voltage readings and one with the current reading of the input file. The output file paths are based on the path specified in the `outputfile` variable and postfixed with *volts* and *amps*, respectively.

C. Common Commands for Offsetting Negative Values

The code in Listing 4 converts a CSV file, the path of which is stored in the variable `inputfile`, into another CSV file, the path of which is stored in the variable `outputfile`, such that there are no negative values in the output file without changing the entropy.

```
1 min=`sort -n "$inputfile" | head -1`
2 awk '{ printf("%.2f\n", $1 - '$min') }' \
3   "$inputfile" > "$outputfile"
```

Listing 4. Offsetting of negative values for CSV files for compression.

In line 1, the smallest value from the input file is stored in the variable `min` through command execution (enclosed by `` ` ` `` characters). This value is obtained by sorting the input file with `sort` numerically (n parameter) and extracting the first line (`1` parameter) of the sorted output with `head`.

In line 2, the minimum value is subtracted from all values of the CSV input file using `awk`. Since the variable `min` which stores the minimum value is negative, this operation results in offsetting each value. More precisely, the first (and only) value in each line (`$1` variable) is offset by the minimum (to a value of zero or greater) and printed with two decimal places using `awk`'s `printf` command and the corresponding format specifier. The output is the content of a CSV file with one column, containing the offset values.

In line 3, this content is redirected to the output CSV file using the > operator. If any of the input files from the pre-processing steps are used, the output file is an offset variation of this input file, containing only positive values, but the same number of decimal places without any change in entropy.

Appendix B. Run Time Results

The methodology from [10] is used to measure the run time of the compression approaches for comparison. The measurements are limited to houses 3 and 5 of the MIT REDD high-frequency raw data set. The same hardware and software are used to enable a meaningful comparison.

Figure 3 shows the average run time per value for all compression approaches described in Section II. The results are shown per channel, with horizontal bars denoting the

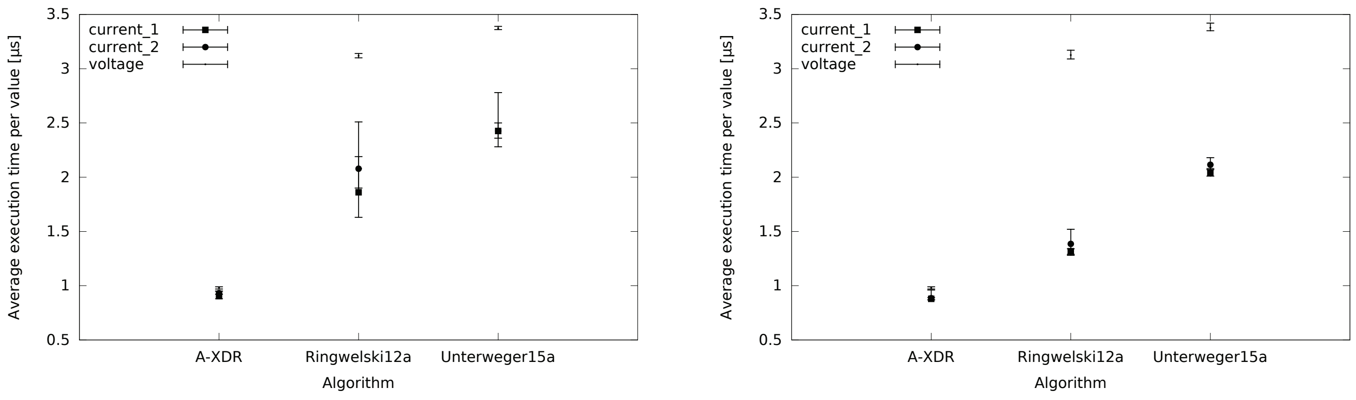

Figure 3. Run time of different algorithms for houses 3 (left) and 5 (right) from the MIT REDD high-frequency raw data set: The results are comparable to the results from [10] for low-frequency consumption data.

minimum and maximum run times, respectively. Three conclusions can be drawn from the results.

First, apart from A-XDR, the channel type – voltage (no marker) or current (filled rectangles and circles) – has an impact on the run time. This is due to the fact that the voltage channels contain larger values, thus taking more time to process. For A-XDR, this effect is practically negligible since the number of operations per input character is small in comparison to LZMH coding (Ringwelski12a [8]) and DEGA coding (Unterweger15a [9]).

Second, there is a performance difference between the two houses, 3 (left) and 5 (right). This shows that different signal characteristics impact the run time per value, although the values only differ little for most channels, e.g., the voltage channels are practically identical in terms of run time for both houses. LZMH coding (Ringwelski12a [8]) is most sensitive to changes in input data characteristics – the run time for the current channels differs by up to 25% between the two houses. This means that special care and additional spare resources are required when using LZMH coding for high-frequency data compression.

Third, the results shown in Figure 3 are comparable to the results from [10] which applied the compression algorithms to low-frequency consumption data. A-XDR consuming around 1 µs per value and DEGA coding (Unterweger15a [9]) consuming around 2 µs for the current channels are practically identical results; only the voltage channels require about 50% more processing time for DEGA coding, with slightly higher differences for LZMH coding (Ringwelski12a [8]). However, all results have the same order of magnitude as in [10].

In summary, high-frequency voltage and current data take roughly the same amount of time per value to compress as low-frequency consumption data. This is an important design consideration for smart meter compression. However, with the data at hand having a 50 kHz sampling rate, the total time required for compression is non-negligible.

Although all channels can be processed (with all algorithms) in real time on a desktop CPU (a maximum of 3.5 µs per value for 50000 values times 3 channels requires 525 ms of run time per second), it is not likely that a smart meter is able to compress 50 kHz data in real time for now. In a few years' time, however, this may be feasible. In contrast, on storage and processing servers, processing capabilities are likely sufficient as of right now. However, further optimizations of the implementations are desirable given the potential scale of the data.