Resumable Load Data Compression in Smart Grids

Andreas Unterweger, *Student Member, IEEE,* and Dominik Engel, *Member, IEEE*

***Abstract*—We propose a compression approach for load profile data, which addresses practical requirements of smart metering. By providing linear time complexity with respect to the input data size, our compression approach is suitable for low-complexity encoding and decoding for storage and transmission of load profile data in smart grids. Furthermore, it allows for resumability with very low overhead on error-prone transmission lines, which is an important feature not available for standard time series compression schemes. In terms of compression efficiency, our approach outperforms transmission encodings that are currently used for electricity metering by an order of magnitude.**

***Index Terms*—Compression, evaluation, load data, resumability.**

I. Introduction

SMART GRIDS rely on information and communication technology to measure, transfer, and manage detailed data on grid status. Smart metering is an important component of this system, providing detailed data in the distribution network. This data forms one of the key components for use-cases in the smart grid, such as energy feedback [1], grid monitoring, and load forecasting [2].

The most important arguments for compressing smart meter data are discussed in detail below: 1) data volume; 2) communication bandwidth; and 3) energy efficiency. Each of these arguments is valid for almost all use-cases of smart metering. However, the degree to which compression is advantageous, depends on the specific requirements of the use-case, such as the volume of data produced in smart metering, the need for (near) real-time transmission, or the predominant direction of communication (while some use-cases, e.g., real-time pricing, push data to the meter, most use-cases involve the meter transmitting data to a data concentrator). Compression is needed most for use-cases which generate a high volume of data, such as monitoring of grid stability, which requires fine-grained data with low delay.

A. Data Volume

While in the traditional billing use-case, the data volume is very small and, therefore traditionally there was no need for data compression whatsoever (even for automated meter reading), it is evident that for the use-cases in the smart grid, data volume increases dramatically: for instance, the data volume of load profile data at a granularity of 1 s and double-precision floating point for the 40 million households in Germany amounts to 25 TB of raw data per day. With the method presented in this paper, this volume can be reduced by nearly 90% to approximately 2.6 TB (assuming data properties similar to the test data). This reduction is not only beneficial in terms of reducing the volume of transmitted data, but also positively impacts storage requirements at the Distribution System Operator.

B. Low Bandwidth

Many smart meters will be connected to low-bandwidth communication links, such as powerline communication (PLC) links. Compression is an important tool to make the best use of the available bandwidth. For example, PLC is more reliable for lower data rates. Through compression, reliability can therefore be increased. Another example for a benefit of compression is the number of communicating parties. In some scenarios a number of smart meters need to communicate within the same network segment, often using collision detection or avoidance. The probability of collision increases with the volume of data each smart meter tries to transmit in the same time interval, up to a point where communication becomes impossible. With compression, the data rate can be reduced and with it the probability of collision. Therefore, compression enables more smart meter to communicate in the same multiple-access segment.

C. Energy Efficiency

The case for data compression of load profiles is also supported from the perspective of energy consumption. The idea of smart grids is closely linked to increasing energy efficiency. Care should be taken for the components of the smart grid to also reflect this endeavor through economical use of energy. The power required for the transmission of bits significantly exceeds the power required for the computational complexity of compression algorithms (e.g., on the *Mica2dot* platform, for the power needed to transmit 1 bit, more than 2000 clock cycles can be executed [3]). Thus, the employment of compression methods saves energy and the effect is multiplied by the vast number of smart meters in the field. The computational capabilities of smart meters will definitely support compression methods such as suggested in this papers (smart meters will need to support at least basic cryptographic primitives [4], which are more demanding than the operations needed for compression as presented here).

Manuscript received March 1, 2014; revised July 2, 2014; accepted October 10, 2014. This work was supported in part by the Austrian Federal Ministry of Science, Research, and Economy, and in part by the Austrian National Foundation for Research, Technology, and Development. Paper no. TSG-00181-2014.

A. Unterweger is with the University of Salzburg, Department of Computer Sciences, Salzburg A–5020, Austria; and also with the Salzburg University of Applied Sciences, Josef Ressel Center for User-Centric Smart Grid Privacy, Security, and Control, Urstein Süd A-5412, Austria.

D. Engel is with the Salzburg University of Applied Sciences, Josef Ressel Center for User-Centric Smart Grid Privacy, Security, and Control, Urstein Süd A-5412, Austria (e-mail: dominik.engel@en-trust.at).

Digital Object Identifier 10.1109/TSG.2014.2364686

Apart from good compression, a method for compressing load data should fulfill a number of other requirements.

1) Low to moderate computational complexity to keep power and processor requirements for smart meters low.
2) Low memory requirements (e.g., the use of large dictionaries makes the smart meter unnecessarily expensive).
3) Low overhead for initialization.
4) *Ability to Resume After Interruption:* If the communication link to a smart metering device is temporarily down, synchronizing the compression algorithm needs to be fast and efficient.

We propose an approach to compression of load profile data that fulfills all of the above criteria.

The rest of this paper is structured as follows. Section II gives an account of related work in the areas of load data representation and time-series compression. Section III describes the characteristics of load data, motivating the design of our proposed compression approach, which is presented in Section IV and analyzed in detail in Section V. In Section VI, our approach is evaluated and compared to existing representations with respect to transmission size and computation time. Finally, we provide an outlook in Section VII before we conclude our paper in Section VIII.

II. Related Work

In [5], standard general-purpose compression algorithms are applied to publicly available load data set. The evaluation shows that load data is well suited for compression. We use the same data set in this paper and conduct a more detailed analysis on compressibility of load data. Using the same data set, we can also show that the approach proposed here is comparable to standardized methods in terms of compression performance, while offering additional features such as resumability, which are important for real-world use.

Compression has been proposed in other areas of the smart grid. The compression of phasor measurement units (PMUs) data is the field most closely related to smart meter readings compression. In [6], different data compression techniques for PMU readings are discussed and evaluated. Ning *et al.* [7] proposed a wavelet-based compression technique for the readings of PMUs. In a similar vein, Khan *et al.* [8] proposed the use of the embedded zerotree method for PMU measurements. While approaches for compressing PMU data are relevant to the subject area considered here, the compression of load data from smart meters differs significantly from PMU readings, by: 1) the origin of the data and consequently the properties of the data; 2) the number of sensors in the field, which is vastly higher in smart metering; and 3) the requirements for practical operation, such as real-time transmission of values.

In the general area of time-series compression, there are a number of contributions, the most notable and active field being audio compression [9]. Another active research area in compression is, of course, centered on video (see [10]). Methods from both fields can be considered for adaptation for load data compression. In fact, some approaches from the area of motion data compression show potential to prove useful when adapted to load data, as will be discussed below.

In practical operation in energy grids, currently no compression is applied by any of the standardized data formats in smart metering. The "open smart grid protocol" [11], which is a protocol suite spearheaded by European Telecommunications Standards Institute (ETSI), defines a format for transferring metering data, using up to 16 channels in the same interval. "All channels are stored as total values (no differential values) [11, p. 34]," and no compression is applied.

The smart metering coordination group, working under EU standardization mandate M441, has defined a functional reference architecture for communications in smart metering systems in a CEN/CENELEC/ETSI technical report [12]. Data model standards and communications profile standards are considered in the report, but data compression is not addressed.

The "device language message specification" and the "companion specification for energy metering" provide data formats and communication standards for automatic meter reading. The relevant standards for the data format are IEC 62056-21 [13] and IEC 62056-53 [14]. A lower layer encoding for metering values, the A-XDR encoding rule, is specified by IEC 61334-6 [15].

III. Load Profile Data Characteristics

Load profiles are time series of electrical power consumption. While the measurement precision is configurable and use-case dependent to a large extent, all load profile data with similar sampling intervals exhibit certain characteristics, some of which will be described in this section.

Note that there may be several other data characteristics which depend on the use-case or are limited to a number of data sets. As we intend to describe general characteristics which apply to a large percentage or even all load profile data in a smart-meter scenario with second- to minute-granularity of sampling, we omit use-case- and data-set-specific characteristics.

On detailed examination of load profiles of consumer households, it can be noticed that, while most consecutive values within a load profile tend to exhibit small value differences between one another, few values exhibit large differences. Depending on the time difference between two consecutive values, this effect is more (e.g., when the sampling interval is in the range of seconds) or less dominant (e.g., when the sampling interval is in the range of minutes or even coarser).

In order to show that this is true for a large number of load profiles, we analyze the properties of consecutive values in a number of load profiles from different data sets. We use the low frequency Massachusetts Institute of Technology (MIT) Reference Energy Disaggregation Data Set (REDD) data set [16] (abbreviated as MIT data set henceforth) as well as the TU Darmstadt tracebase data set [17] (abbreviated as TUD data set henceforth).

The MIT data set consists of a total of 116 load profiles. Each load profile contains average power readings of one individual circuit from one of six houses. The data is sampled in intervals of 1 s with a precision of 0.01 watts, i.e., the

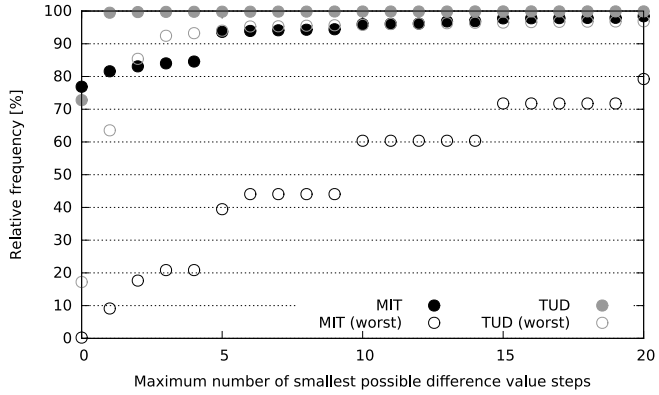

Fig. 1. Frequency of small consecutive value differences in the MIT (black) and TUD (gray) data sets. Filled circles denote the average relative frequency over all load profiles, while empty circles accentuate the load profile with the smallest relative frequency.

smallest nonzero difference between two consecutive values is 0.01 watts.

The TUD data set consists of a total of 1836 load profiles. Each load profile contains average power readings of one of 44 electric appliances. Like the MIT data, the data is sampled in intervals of 1 s with a precision of 1 watt, i.e., the smallest nonzero difference between two consecutive values is 1 watt.

Fig. 1 shows the relative frequency of the 20 smallest possible absolute value differences between consecutive values for both the MIT and the TUD data set. For example, 93% of all value differences in the load profiles of the MIT data set (illustrated by filled black circles) are between -0.05 and 0.05 watts (both inclusive), corresponding to the five smallest possible value differences. However, there is at least one data set (illustrated by empty black circles) in which only 40% of all value differences are within these limits.

The plots clearly show that the biggest part, i.e., more than 95%, of the consecutive value differences are between -0.1 and 0.1, corresponding to only ten of the smallest possible value steps. This is somewhat surprising considering that the maximum absolute value difference in the load profiles from MIT data set is as large as 6680.55 watts, corresponding to 6 68 055 value steps.

The distribution of the value differences in the TUD data set (gray filled circles) is even more surprising. Quasi all, i.e., more than 99.99%, of the consecutive value differences are -1, 0, or 1, although the absolute value differences are as large as 4879 watts maximum.

In both, the TUD and the MIT data set, less than the smallest 0.02% possible value differences (with respect to the maximum occurring value difference in all load profiles) make up more than 99% of all differences. Although, this is an average value summarizing all load profiles of the respective data set, it clearly shows that the load profile data tends to exhibit very small changes between two measurement points with respect to the corresponding data values.

As there is a high amount of load profiles per data set, the empty circles in Fig. 1 depict the characteristics of the load profiles with the lowest relative frequency of small value differences per data set for both, the MIT (black) and the TUD data set (gray).

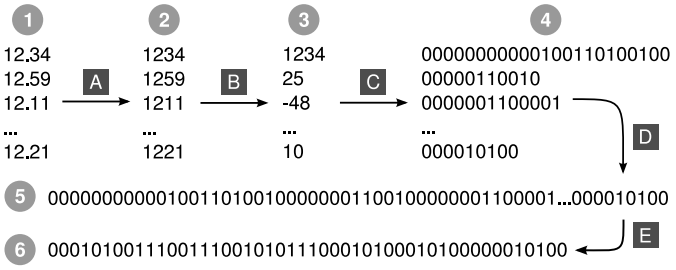

Fig. 2. Proposed compression approach: a list of values from a load profile (1) is transformed to a compressed bit string (6) through five successive encoding steps (A–E).

In the TUD data set, the worst case scenario in terms of the relative frequency of value differences of no more than 6 watts is one data set with 95% relative frequency. In other words, only 5% of all value differences in the worst case data set are larger than six. Similarly, in the worst case load profile in the MIT data set, there are only a little more than 20% of all value differences whose absolute value is greater than 20.

While both data sets contain load profiles where the number of small value differences is significantly smaller than on average, i.e., smaller than for all load profiles of the respective data set, the percentage of small value differences is very high and increases significantly with every additional value difference step.

Considering that the displayed value difference steps in Fig. 1 only cover about 0.003% (20 out of 6 68 055) and about 0.41% (20 out of 4879) of the actual value difference range of the MIT and TUD data set, respectively, this allows for the following conclusions on load profile data with second-granularity, thereby summarizing some of their compression-relevant characteristics.

1) Quasi all value differences of two consecutively sampled load profile data values are very small with respect to the possible range of value differences.
2) Large value differences are very rare, even in worst-case load profiles (in the analyzed data sets).

IV. Proposed Compression Approach

We propose a compression approach consisting of five steps (denoted as A–E) illustrated in Fig. 2, which exploits the load profile data characteristics described in Section III. Our algorithm takes a list of values from a load profile (1) as input and outputs a compressed binary representation of it (6). In the following, the five processing steps (A–E) are described.

A. Normalization

Floating point operations typically accumulate rounding errors due to their finite precision, which may lead to undesirable side effects. Furthermore, floating point operations are often more expensive in terms of computation time than their integer counterparts due to the lack of floating point units in most embedded systems [18]. Therefore, the first step of our approach (denoted as A in Fig. 2) is the conversion of floating point values to integer values, referred to as normalization.

As the precision of each value is bounded for both, technical and physical, reasons, so is the precision of a list of

values. Let p_i denote the precision of the ith value v_i, expressed in the number of decimal places after the decimal point. Consequently, the precision of each value contained in the list of values is bounded by

$$p_{\max} = \max_i p_i. \quad (1)$$

Moving the decimal point of each value by $p_{\max}$ decimal places to the left, all values can be expressed as normalized integer values n_i with the same precision

$$n_i = v_i \cdot 10^{p_{\max}}. \quad (2)$$

This calculation is illustrated for $p_{\max} = 2$ as step A in Fig. 2. We observed that $p_{\max}$ is typically identical for all input values v_i for quasi all real-world load profile data since the measurement precision hardly ever changes.

Note that the normalization step may be omitted if the input values have no decimal places after the decimal point.

B. Differential Coding

As described in detail in Section III, the differences between consecutive values are mostly very small. This property can be exploited by differential coding, i.e., by storing only the differences between two consecutive values instead of the actual values. Note that this is closely related to differential pulse-code modulation [19].

The differential coding in our approach is illustrated as step B in Fig. 2. The differential values d_i are calculated from the normalized input values n_i from the previous step by a simple subtraction for all values but the very first

$$d_{i>0} = n_i - n_{i-1}. \quad (3)$$

The first value remains unchanged, since there is no reference value for it to be subtracted from

$$d_0 = n_0. \quad (4)$$

C. Variable Length Coding

In order to actually exploit the fact that a large number of difference values d_i are likely to be small (see Section III), a variable length code is needed. We use a zeroth order signed exponential-Golomb code as used in the H.264 standard [10] and described in detail in [20].

Although exponential-Golomb codes are optimal for geometrically distributed values [20] and the difference values d_i are unlikely to be exactly distributed in this way, we use this code as it is able to compactly represent small difference values, which are very likely to occur. Large difference values, which are not very likely to occur, may slightly affect coding efficiency.

Table I shows both signed and unsigned zeroth-order exponential-Golomb code words for the corresponding integer input values, i.e., possible d_i in our use-case. Note that it is necessary to use signed exponential-Golomb code words since the difference values d_i may be negative.

A value of zero can be encoded using just one bit. All other signed exponential-Golomb code words can be constructed by mapping the unsigned exponential-Golomb code words alternately to negative and positive values, respectively. Using this encoding, each difference value d_i is transformed into a corresponding variable length code word c_i as illustrated as in Fig. 2 (step C).

TABLE I
LIST OF EXEMPLARY VALUES AND THEIR RESPECTIVE SIGNED AND UNSIGNED ZEROTH ORDER EXPONENTIAL-GOLOMB CODE WORDS. HYPHENS DENOTE INVALID VALUES. ADOPTED FROM [21]

Value	Unsigned Exp.-G. code word	Signed Exp.-G. code word
...	–	...
-4	–	0001001
-3	–	00111
-2	–	00101
-1	–	011
0	1	1
1	010	010
2	011	00100
3	00100	00110
4	00101	0001000
...	...	...

D. Code Word Concatenation

To group the variable length code words c_i from the previous step for the subsequent step, they are concatenated to a bit string b as illustrated in Fig. 2 (step D). Note that no delimiters are required since the code words include implicit length information (the number of leading zero bits is equal to the number of value bits after the delimiting one bit).

E. Entropy Coding

As a final step, we perform entropy coding on the concatenated bit string b in order to get the final compressed bit representation e, as illustrated in step E in Fig. 2. By using an arithmetic coder, which theoretically allows perfect, i.e., zero-redundancy entropy coding under certain conditions [22], this aims at removing most of the remaining redundancy.

Since the code words c_i have variable length and are potentially large, we use binary arithmetic coding which operates on bit level and therefore only distinguishes two symbols—zero and one. Since the probabilities of these two symbols, which are required as an input for the arithmetic coder, may differ depending on the input values, we start with 50:50 probabilities and perform adaptive encoding, i.e., we modify the probabilities of the two symbols during the encoding process according to their actual occurrences.

In order to avoid floating point operations during arithmetic coding, an implementation relying only on integer operations is recommended. As described in Section VI, we use an implementation which is based on the algorithm proposed in [22]. Although, it is possible to faster implementations, see [20] or [23], the latter rely on approximations. Therefore, their final bit string length may in some cases be slightly different from our results.

F. Summary

As summarized in Fig. 2, the input values v_i (1) are normalized to integer values n_i (2), which are differentially coded as d_i (3). After each differentially coded value d_i is encoded as

TABLE II
WORST-CASE TIME COMPLEXITY OF OUR APPROACH AND ITS CORE STEPS WITH RESPECT TO THE NUMBER OF INPUT VALUES n AND THEIR MAXIMUM SIZE m IN BITS

Step	Per value	Combined
Differential coding	$O(m)$	$O(mn)$
Variable length coding	$O(m)$	$O(mn)$
Entropy coding	–	$O(mn)$
Combined	–	$\boldsymbol{O(mn)}$

TABLE III
WORST-CASE SPACE COMPLEXITY OF OUR APPROACH AND ITS CORE STEPS WITH RESPECT TO THE NUMBER OF INPUT VALUES n AND THEIR MAXIMUM SIZE m IN BITS

Step	Per value	Combined
Differential coding	$O(m)$	$O(m)$
Variable length coding	$O(m)$	$O(m)$
Entropy coding	–	$O(1)$
Combined	–	$\boldsymbol{O(m)}$

one variable length code word c_i (4), all code words are concatenated to a single bit string b (5), which is finally entropy coded as a compressed bit string e (6).

G. Decoding Process

Decoding a bit string encoded with our approach requires applying the inverse operations in the reverse order, i.e., the decoding equivalents of the encoding steps (E–A). These decoding steps are shortly described below.

First, entropy decoding (inverse of step E) is performed on the bit string, yielding a string of variable length code words, which can be split (inverse of step D) due to the implicit length information they contain (see Section IV-D). The exponential-Golomb code words are then decoded (inverse of step C) to yield difference values. Finally, these values are added to their respective predecessors (inverse of step B) in order to get the original absolute values, which can be denormalized through division (inverse of step A).

V. ALGORITHMIC PROPERTIES

The approach presented in Section IV exploits the characteristics of load profile data for compression. However, its applicability for a smart metering use-case is not obvious. Hence, this section analyzes its properties with a strong focus on practicality. It describes those features which are relevant when the proposed approach is used to process and transmit load profile data and provides a detailed overview of its time and space complexity.

A. Algorithmic Complexity

When encoding n values with a maximum size of i and m bits each before and after normalization, respectively, all subsequent steps of our proposed compression approach require processing time and memory depending on m, n or both. In this section, we derive the worst-case time and space complexity of our approach and the aforementioned steps. The results are summarized in Tables II and III.

We use asymptotic notation [24] and assume that all values are available in memory when they are needed by our algorithm, i.e., they are either all in memory the whole time, consuming $O(mn)$ bits, or loaded into memory one by one on demand, consuming $O(m)$ bits. Furthermore, we assume that the encoded data is either transmitted or stored immediately so that no temporary memory for the fully encoded bit representation has to be taken into account and the algorithm's space complexity analysis can focus on the overhead of the algorithm itself.

The first step, i.e., the normalization, largely depends on the format of the input values. If they are already integer, no operation needs to be performed. Otherwise, one multiplication for the i-bit input values, followed by an optional rounding operation, is required. This requires $O(i^2)$ time complexity when using a straight-forward implementation of an i-bit multiplication algorithm [24]. For the optional rounding operation, the same applies.

Since the normalization step itself is optional and, if performed, highly dependent on the input values and their format, its time and space complexity are not included in Tables II and III, respectively. It should be noted, however, that the space complexity of the multiplication is $O(i)$. If i is proportional to m, this is equal to $O(m)$.

The second step, i.e., the differential coding, can be performed value by value and requires only the last value to be stored in order to calculate the current value difference. This takes up m bits of space constantly, plus m bits for the calculated difference, being of a total space complexity of $O(m)$.

One m-bit subtraction per value has a time complexity of $O(m)$ [24]. Since $n-1$ values need to be processed, this step is performed $n-1$ times, corresponding to $O(n)$ time complexity times the complexity per value, totaling a time complexity of $O(mn)$ for all values.

The third step, i.e., the variable length coding, can also be performed value by value. Since the worst-case encoding, i.e., the variable length encoding of the largest possible value, is proportional to the input value bit size m [20], the total space complexity of this step is $O(m)$.

Calculating a variable code word of an i-bit value requires a constant number of additions and subtractions as well as i divisions or shift operations [20], followed by a maximum of $2i+1$ bit writing operations. With the worst-case input bit length being proportional to m, this requires a time complexity of $O(m)$, since m-bit additions, subtractions and shift operations are all of time complexity $O(m)$ [24]. In total, this yields a time complexity of $O(mn)$ for all values.

Note that the fourth step, i.e., the bit string concatenation, is not listed in Tables II and III. This is due to the fact that the output values of the preceding step can be passed directly to the next step, i.e., the entropy coding step, one by one so that no intermediate memory is required and no actual concatenation has to be performed. The bit string concatenation can therefore be regarded as a conceptual step

rather than an actually necessary one in a straight-forward implementation.

The fifth and final step, i.e., entropy coding, is performed on the whole output of the third step, value by value, one bit at a time. This conceals the intermediate concatenation step as explained above. Since the output of the third step has a total length which is $O(mn)$, the per-bit time complexity of the entropy coder times mn yields the time complexity of the complete entropy coding step.

In total, the time complexity of an adaptive binary arithmetic encoder for each input bit is constant, i.e., of time complexity $O(1)$ with respect to m and n [22]. Hence, processing $O(mn)$ input bits is of time complexity $O(mn)$. The space complexity is constant, i.e., $O(1)$ [22].

Summing up the space and time requirements of all described steps, this yields a total time complexity of $O(mn)$, which is proportional to the number of input values and their maximum size in bits and thus equal to, e.g., the time complexity of performing n m-bit additions, i.e., relatively low. The space complexity is $O(m)$ and therefore not dependent on n, which enables encoding with very modest space requirements.

Since, decoding involves the inverse operations of the described encoding steps in opposite order (as explained in Section IV-G), the time and space complexity of a decoder are expected to be equal to the encoder's. In order to avoid a complete complexity analysis of the decoding process, we refer to the symmetry of the operations involved to claim this without explicit proof.

B. Resumability

As load profile data is transmitted extensively in smart grids, the adequacy of our compression approach for this use-case has to be assessed. Although effective compression reduces data size and thus transmission time, it may have undesirable side effects compared to uncompressed transmission, for example when data is lost.

Although our approach does not include native error detection capabilities, it allows for retransmissions of parts of the data with very low overhead. This subsection discusses the conditions under which a lost part of the transmitted data can be retransmitted so that the transmission process can be resumed. Furthermore, the retransmission procedure as well as its overhead are discussed.

The state of the decoder during the decoding process is limited to a small number of variables. First, the differential coding step stores one m-bit variable containing the last coded value, as described in Section V-A. Second, the arithmetic coder stores three machine-word-sized integers representing the probability of the symbol zero, the current interval and the number of bits to be output after the next one, respectively.

Since this information is sufficient to represent the entire state of the decoder, a decoding process can be resumed by sending the aforementioned variables. For example, if $m = 32$ and the machine word size is 16, the decoder state can be represented by $32 + 3 \cdot 16 = 32 + 48 = 80$ bits, or 10 bytes.

Since our compression approach is not able to detect errors, it relies on an encapsulation format or transmission protocol to do so. In case of an error, the decoder's state can be reset to the last known good state by keeping a copy of the decoder's state after each successfully decoded data packet. The retransmission overhead is then equal to the size of the decoder state, e.g., 10 bytes.

Alternatively, it is possible for the decoder to request the encoder's state in order to resynchronize. Since both perform symmetric operations, their states have to be equal. Note that they can only exchange their states if the used protocol and channel allow them to do so.

This allows for resumability, i.e., the possibility to resume the decoding process at a given point. However, it is not possible to reconstruct preceding values which have been omitted between the last known good value and the resumed one (if there are any), since prior decoder states cannot be reconstructed. However, it is possible to deliberately omit parts of the data as long as the decoder state required to start decoding after the omitted parts is available.

If our approach is used in combination with packet-based transmission, we suggest to add the decoder state to each packet. This way, the decoder can process each packet independently and does not rely on the retransmission of preceding packets in order to decode the current one. The overhead should be negligible for typically large packet sizes, e.g., 1500 bytes for IEEE 802.3 (Ethernet), as well as for practical values of m and typical machine word sizes.

VI. Performance Evaluation

In order to evaluate the performance of our proposed approach, we analyze its compression efficiency as well as its processing time for a number of data sets and compare the results to those of related standards. Before discussing the results, we describe our implementation and test environment as well as the used data sets and the related standards used for comparison.

A. Related Standards

As mentioned above, we use two common uncompressed data formats for comparison with the proposed method. We only consider the encoding of the actual payload, i.e., we omit any encapsulation and/or protocol overhead since this would bias the results.

The first uncompressed format is described in IEC 61334-6 [15], which is also referred to as A-XDR encoding. Although this standard describes a number of possible encodings for numerical values, we consider the fixed-length unsigned integer encoding [15, Section VI-A1a] to be the most practically relevant one due its low per-value overhead.

As explained above, we omit all encapsulating identifier and length fields to make the comparison between the our encoding and the one from A-XDR as fair as possible. This way, n encoded values with a maximum of m bits size each require a constant amount of $n \cdot \lceil m/8 \rceil$ bytes or $8n \cdot \lceil m/8 \rceil$ bits, as illustrated in Fig. 3(a).

Since A-XDR provides no explicit encoding for floating point values, all input values to the A-XDR encoding process are considered to be integer values. This can be assured

Byte offset	0	1
Payload	00000000	1111011

(a)

Byte offset	0	1	2	3
Payload	00110001	00110010	00110011	00000000
ASCII rep.	'1'	'2'	'3'	'\0'

(b)

Fig. 3. Encodings of the integer value 123 from related standards. (a) 16-bit fixed-length unsigned integer A-XDR encoding. (b) ASCII-based IEC 62056-21 encoding with trailing zero (delimiter).

through a normalization preprocessing step, as described in more detail in Section VI-B. As this preprocessing step is part of our proposed compression algorithm anyway, it can be performed once on the input data instead of once within each algorithm. This also allows for a fairer comparison of all evaluated encodings since they are provided with the same input data.

The second uncompressed format is described in IEC 62056-21 [13]. It encodes values in data blocks [13, Section VI-C4] which consist of one or more data lines [13, Section VI-E1], which themselves contain one actual value and its corresponding unit each. All data lines consist of a limited set of printable characters expressed in the ASCII character set [25].

Again, we only consider the actual payload, i.e., the ASCII-encoded values and omit the units and other protocol overhead. Note, however, that one additional byte per value is required to separate consecutive values due to their variable length, as illustrated in Fig. 3(b). This is not necessary for our approach or the A-XDR encoding described above since the latter uses a fixed number of bytes and our approach implicitly encodes length information (see Section IV-D).

B. Implementation and Test Environment

We implemented our approach proposed in Section IV in Python. For the arithmetic coding step, we used D. MacKay's implementation[1] which is practically identical to the implementation from [22]. Since MacKay's implementation operates on a bit string, the preceding step of our approach, i.e., the exponential-Golomb coding, outputs such a bit string instead of the corresponding byte sequence, as opposed to all prior steps.

In order to make the comparison between our approach and the ones from related standards fair, we reimplemented the latter so that they output bit strings as well. Note that, although this alters the processing time slightly, it still allows comparing the algorithms with respect to the order of magnitude of processing time.

As described in Section VI-A, some of the algorithms from related standards are not capable of handling noninteger (i.e., in this case, floating point) values. To simplify processing and make the processing time comparison fairer, all data is preprocessed by applying the normalization step described in Section IV-A. Thus, all algorithms operate on integer input data. This reduces the processing time of those algorithms in need of preprocessing, but still allows for a comparison with respect to the order of magnitude of processing time.

[1] http://www.inference.phy.cam.ac.uk/mackay/python/compress/#AC

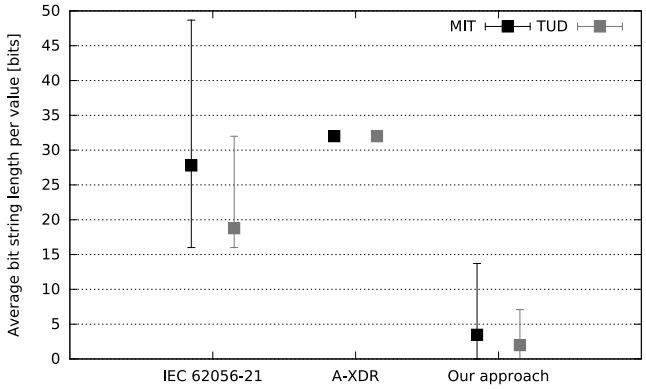

Fig. 4. Average value size in bits for the MIT (black) and the TUD data set (gray). The bars indicate the average number of bits per value for those load profiles which require the minimum and maximum number of bits in each data set, respectively.

Our test environment is a server hosting an Intel Xeon E5-2620 CPU with six physical cores running at 2 GHz each. The server runs Ubuntu 12.04.2 LTS on a 64-bit Linux 3.2.0-48 kernel. We use Python 2.7.3 and its built-in clock function from the time module to measure processing times.

C. Data Sets

For our evaluation, we use the MIT and TUD data sets described in Section III. As the values in the MIT data set are stored with a precision of 0.01 watts, we normalize them by multiplying them by 100 as described in Section IV-A. The values in the TUD data set are already stored as integers and therefore require no normalization.

For the fixed-length A-XDR encoding we used a bit length of 32 bits, since this is the smallest whole-byte size which is a power of two (which is typically used in practice) that allows covering the whole value range present in the input files. Similarly, our algorithm uses 32 bit variables for the difference calculation in the differential coding step (see Section IV-B).

D. Compression

For all load profiles of each data set, we evaluate the average number of bits required to represent one value. Fig. 4 illustrates this for all tested approaches. Obviously, fixed length A-XDR coding always requires 32 bits, while the IEC 62056-21 encoding and our approach requires a variable number of bits.

It is clear that our approach outperforms the other two by an order of magnitude. Furthermore, there is no load profile for which our approach is inferior to one of the other two, since the maximum number of bits per value required by our approach is always significantly smaller than the minimum amount of bits per value required by both, the fixed-length A-XDR and the IEC 62056-21 encoding.

As the value range of the TUD data set is smaller than the one of the MIT data set (see Section IV-B), the average number of bits required per value is significantly smaller for the TUD

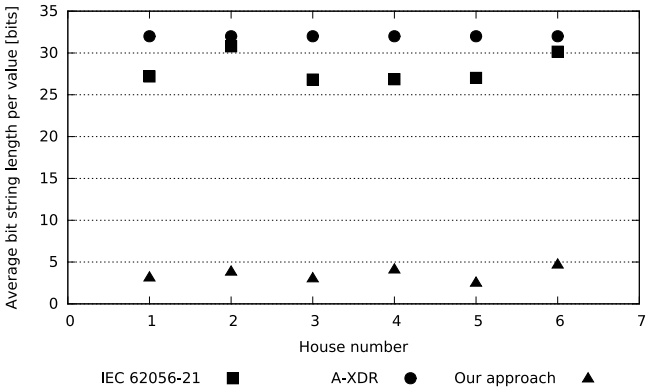

Fig. 5. Average value size in bits for each house from the MIT data set. The bars indicate the average number of bits per value for those load profiles which require the minimum and maximum number of bits in each house, respectively.

data set. This reflects in the results for both, our approach and the IEC 62056-21 encoding, which are variable-length codes.

Since the MIT data set is a collection of load profiles from multiple houses, it is possible to investigate the differences between them in terms of compression efficiency. Fig. 5 shows the average number of bits required per value and house for each encoding approach.

Again, our proposed compression approach outperforms the other two by an order of magnitude for each individual house, revealing that averaging the values per data set (as shown in Fig. 4) did not conceal any inefficiencies of our approach. Interestingly, the compression efficiency of our approach shows a slight correlation with the IEC 62056-21 encoding, albeit inherently not proportional.

As the load profiles of each house from the MIT data set correspond to one channel each, it is possible to investigate their individual compression performance. Fig. 6 shows the average bit length per value for each channel of each house. Note that the channel labels are taken directly from the MIT data set's companion files and have not been modified, i.e., corrected orthographically.

For almost all individual channels, i.e., load profiles, our compression approach is significantly superior to the other two. Although, it is not always more efficient by an order of magnitude, it is at least twice as efficient for all channels.

One advantage of our approach becomes clear for channels which are typically constant or quasi constant over long periods of time, e.g., the oven and stove channels in house 1 (top left). Since the load on such a channel typically changes rarely and rapidly, the number of zero and small differences (due to noise and measurement inaccuracies) is very high, allowing our approach to effectively compress the data.

However, channels with relatively unpredictable load behavior, like the mains of all houses, can still be compressed very efficiently as compared to the other two encodings. The same is true for channels with low variable load, like the kitchen outlets of all houses, showing that our proposed approach is able to compress these types of load profiles efficiently as well.

Although, our approach achieves better compression performance for channels with relatively small changes between values, the employed exponential-Golomb code and arithmetic coding still allow for sufficiently good compression in cases of larger changes between values. This is mainly due to the fact that the length of exponential-Golomb codes (in bits) only increases logarithmically with increasing input values, i.e., increasing value differences in our method. Thus, only very large value differences would generate notably longer codewords and thereby impact the compression performance significantly, which is a practical feature of our proposed approach.

E. Processing Time

Fig. 7 shows the average number of microseconds required to encode one value with each approach for the MIT (black) and the TUD data set (gray), respectively. As explained in Section VI-B, our implementation only allows assessing the order of magnitude of the processing times.

Thus, it cannot be asserted that our approach is significantly faster than the other two. However, it requires about the same order of magnitude in terms of processing time per encoded value, hence being comparable to both, fixed-length A-XDR and IEC 62056-21 encoding.

Since all variable-length coding approaches depend on the actual size of their input values, both, our approach and the IEC 62056-21 encoding, clearly require less time per value on average for the TUD data set than they do for the MIT data set. This is clear, since the latter's value range is larger. This confirms the linear dependency of both approaches to the input bit length, allowing for faster processing of small input values.

Note that further investigations, e.g., per house or per channel of the MIT data set, are not meaningful due to the limited measurement accuracy. This is supported by the difference between the average processing times of the MIT and the TUD data set for the A-XDR encoding, which should be zero in theory assuming a perfect implementation and execution environment, since A-XDR is a fixed-length code.

VII. Future Work

Our approach has been shown to perform very well in terms of compression efficiency, but there is still room for improvement. For example, signal characteristics like periodicity could be exploited as most load profiles tend to exhibit weekly, daily or even hourly patterns based on the types of appliances and users. By forming a prediction signal from the last week, day or hour, respectively, one could encode the difference between the current data values and the predicted ones instead of encoding consecutive value differences, thereby further increasing compression efficiency. It should also be possible to combine these two approaches.

Another issue to be addressed is the diversity of the evaluated load profiles. Although the amount and diversity of load profiles from the MIT and the TUD data set are already very high, it would be desirable to have a larger amount of real-world test data available in order to extend the results of this paper. This does not only include data from different sources,

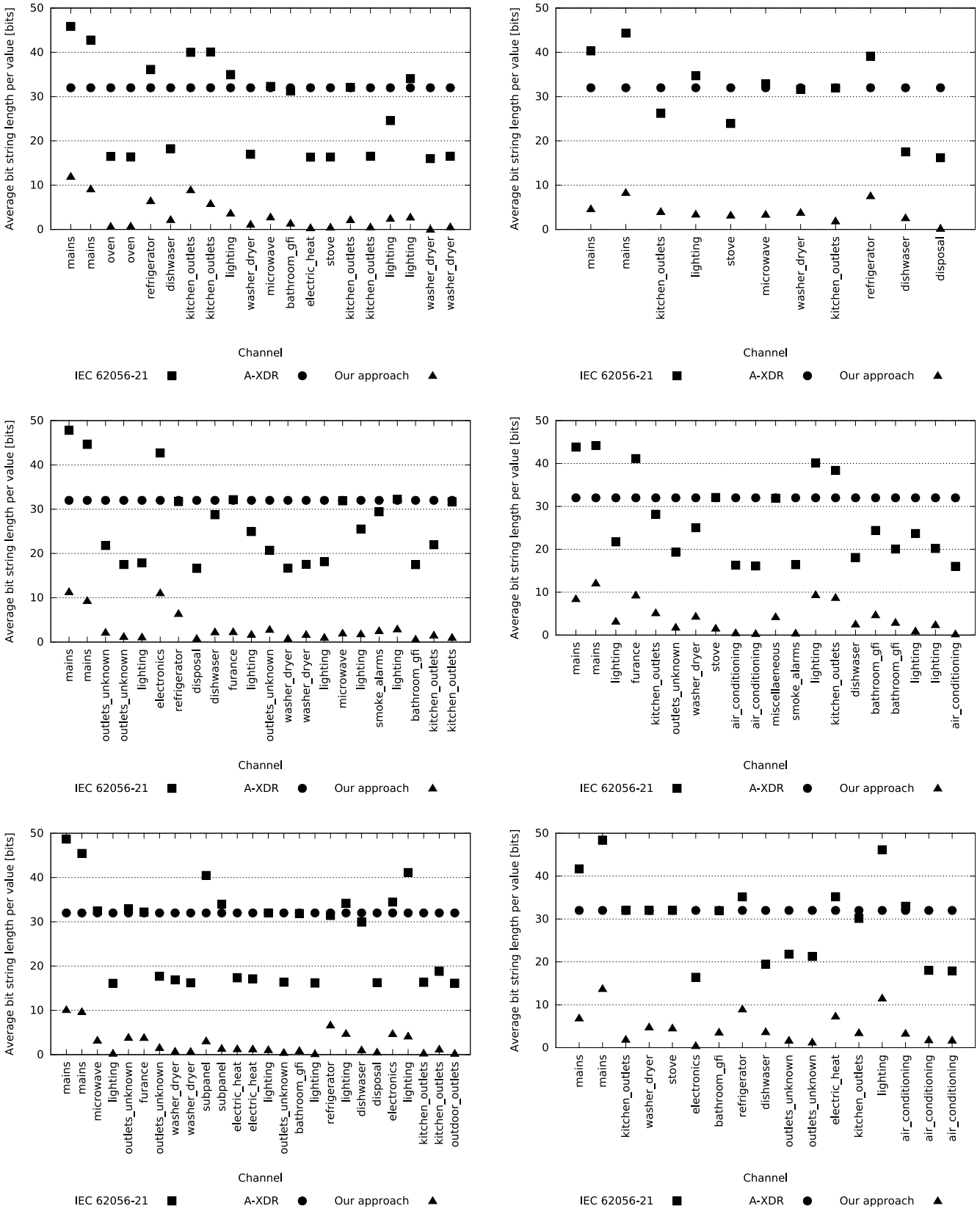

Fig. 6. Average value size in bits for each channel of each house from the MIT data set (top left: house 1; top right: house 2; etc.). Each channel corresponds to one load profile.

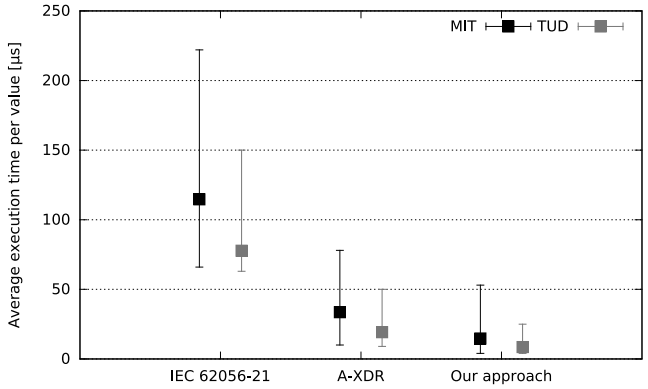

Fig. 7. Average encoding time per value for the MIT (black) and the TUD data set (gray). The bars indicate the average number of microseconds per value for those load profiles which require the minimum and maximum time in each data set, respectively.

but also different granularity, e.g., a sampling time of minutes or even hours. This way, the compression efficiency could be evaluated more thoroughly.

Finally, the overhead introduced by using our proposed resumability feature has to be evaluated thoroughly. This does not only include analyzing the overhead for different loss rates, but for different link types and protocols as well. Since such an analysis is beyond the scope of this paper, it remains future work.

VIII. Conclusion

We proposed a compression approach tailored for the requirements of load profile data transmission in smart metering. We showed that our approach allows for resumability with very low overhead, which enables it to operate in error-prone transmission lines in smart grids. Even with providing resumability, our approach has been shown to maintain the same compression results as standard compression algorithms, which do not provide this important feature. Currently employed state-of-the-art transmission encodings are outperformed by an order of magnitude in terms of compression performance without significantly impacting the processing time required for the encoding process. In summary, the proposed approach is ideally suited for compression of smart meter load data as it delivers competitive compression results with low computational complexity, low memory requirements, low overhead for initialization and the ability to resume after interruption.

References

[1] Z. Fan *et al.*, "Smart grid communications: Overview of research challenges, solutions, and standardization activities," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 1, pp. 21–38, Apr. 2013.

[2] M. Ghofrani, M. Hassanzadeh, M. Etezadi-Amoli, and M. Fadali, "Smart meter based short-term load forecasting for residential customers," in *Proc. North Amer. Power Symp. (NAPS)*, Boston, MA, USA, 2011, pp. 1–5.

[3] A. Wander, N. Gura, H. Eberle, V. Gupta, and S. Shantz, "Energy analysis of public-key cryptography for wireless sensor networks," in *Proc. 3rd IEEE Int. Conf. Pervasive Comput. Commun. (PerCom)*, Kauai Island, HI, USA, 2005, pp. 324–328.

[4] P. McDaniel and S. McLaughlin, "Security and privacy challenges in the smart grid," *IEEE Security Privacy*, vol. 7, no. 3, pp. 75–77, May/Jun. 2009.

[5] 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 *Proc. 2012 IEEE Int. Energy Conf. Exhibit. (ENERGYCON)*, Florence, Italy, pp. 935–940.

[6] J. Kraus, P. Stepan, and L. Kukacka, "Optimal data compression techniques for smart grid and power quality trend data," in *Proc. 2012 IEEE 15th Int. Conf. Harmonics Qual. Power (ICHQP)*, Hong Kong, pp. 707–712.

[7] J. Ning, J. Wang, W. Gao, and C. Liu, "A wavelet-based data compression technique for smart grid," *IEEE Trans. Smart Grid*, vol. 2, no. 1, pp. 212–218, Mar. 2011.

[8] J. Khan, S. Bhuiyan, G. Murphy, and M. Arline, "Embedded zerotree wavelet based data compression for smart grid," in *Proc. 2013 IEEE Ind. Appl. Soc. Ann. Meeting*, Lake Buena Vista, FL, USA, pp. 1–8.

[9] N. Harada, Y. Kamamoto, T. Liebchen, T. Moriya, and Y. A. Reznik, "The MPEG-4 audio lossless coding (ALS) standard—Technology and applications," in *Proc. Audio Eng. Soc. Conv. 119*, New York, NY, USA, Oct 2005, Art. ID 6589.

[10] *Information Technology—Coding of Audio-Visual Objects—Part 10: Advanced Video Coding*, ISO/IEC Standard 14496-10:2009, 2005.

[11] *Open Smart Grid Protocol (OSGP)*, ETSI GS OSG 001 V1.1.1, 2012. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/OSG/001_099/001/01.01.01_60/gs_osg001v010101p.pdf

[12] Smart Meters Coordination Group, "Functional reference architecture for communications in smart metering systems," CEN/CLC/ETSI/TR, Tech. Rep. 50572, Dec. 2011. [Online]. Available: ftp://ftp.cencenelec.eu/EN/EuropeanStandardization/HotTopics/SmartMeters/CEN-CLC-ETSI-TR50572%7B2011%7De.pdf

[13] *Electricity Metering—Data Exchange for Meter Reading, Tariff and Load Control—Part 21: Direct Local Data Exchange*, IEC Standard 62056-21, 2002.

[14] *Electricity Metering—Data Exchange for Meter Reading, Tariff and Load Control—Part 53: COSEM Application Layer*, IEC Standard 62056-53, 2002.

[15] *Distribution Automation Using Distribution Line Carrier Systems—Part 6: A-XDR Encoding Rule*, IEC Standard 61334-6, 2000.

[16] J. Kolter and M. Johnson, "REDD: A public data set for energy disaggregation research," in *Proc. Workshop Data Min. Appl. Sustain.*, San Diego, CA, USA, 2011, pp. 1–6.

[17] A. Reinhardt *et al.*, "On the accuracy of appliance identification based on distributed load metering data," in *Proc. 2nd IFIP Conf. Sustain. Internet ICT Sustain. (SustainIT)*, Pisa, Italy, 2012, pp. 1–9.

[18] J. Ganssle, *The Art of Designing Embedded Systems*. Amsterdam, The Netherlands: Elsevier Sci., 2008.

[19] C. C. Cutler, "Differential quantization of communication signals," U.S. Patent 2 605 361, July 29, 1950.

[20] D. Marpe, H. Schwarz, and T. Wiegand, "Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 620–636, Jul. 2003.

[21] D. Engel, A. Uhl, and A. Unterweger, "Region of interest signalling for encrypted JPEG images," in *Proc. 1st ACM Workshop Inf. Hiding Multimedia Security (IHMMSEC)*, Montpellier, France, 2013, pp. 165–174.

[22] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Commun. ACM*, vol. 30, no. 6, pp. 520–540, Jun. 1987.

[23] W. B. Pennebaker, J. L. Mitchell, G. G. Langdon, Jr., and R. B. Arps, "An overview of the basic principles of the Q-Coder adaptive binary arithmetic coder," *IBM J. Res. Develop.*, vol. 32, no. 6, pp. 717–726, Nov. 1988.

[24] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction To Algorithms*, 2nd ed. Cambridge, MA, USA: MIT Press, 2001.

[25] *Coded Character Sets—7-Bit American National Standard Code for Information Interchange (7-Bit ASCII)*, ANSI Standard X3.4, 1986.

Andreas Unterweger (S'13) received the Diploma degree in information technology and systems management (with distinction) from the Salzburg University of Applied Sciences, Puch/Salzburg, Austria, in 2008, and the Master's degree in computer science (with distinction) from the University of Salzburg, Salzburg, Austria, in 2011, where he is currently pursuing the Ph.D. degree in selective video encryption.

He is an External Lecturer with the Salzburg University of Applied Sciences for the bachelor's degree program. His current research interests include real-time image and video compression, as well as selective video encryption.

Dominik Engel (S'06–M'08) received the Ph.D. degree in computer science from the University of Salzburg, Salzburg, Austria in 2008.

He was a Researcher with the University of Bremen, Bremen, Germany, and the University of Salzburg, and a Product Manager with Sony DADC, Anif, Austria, where he was responsible for video content security. Since 2010, he is a Professor with the Salzburg University of Applied Sciences, Puch/Salzburg, Austria, where he is the Head of the Josef Ressel Center for User-Centric Smart Grid Privacy, Security, and Control. His current research interests include smart grid security and privacy, multimedia security, and technological methods for enhancing end-user trust.