Powerstrip: High-Performance Compression for Energy Data

John R. Ward[*]
Okta, Inc.
jack.ward@okta.com

Sean K. Barker
Bowdoin College
sbarker@bowdoin.edu

ABSTRACT

The proliferation of smart outlets and meters with submetering capabilities has led to an explosion in the availability of device-level energy data. The increasing volume of current and historical data presents a storage and distribution challenge, particularly for utilities and large-scale energy datasets. To address these challenges, we present Powerstrip, a fast, effective, and nearly-lossless compression algorithm for integer energy data. Powerstrip is optimized for device-level measurements and exploits common characteristics of real-world energy consumption to achieve typical compression rates over 90% on such data. We evaluate Powerstrip on real-world energy data and compare against multiple state-of-the-art compression algorithms. Our experiments show that when compared to the best reference algorithms, Powerstrip achieves the highest compression ratios (by up to 35%) as well as the fastest speeds (by up to 70%). We also present case studies demonstrating the potential of Powerstrip for large-scale energy data storage and distribution.

CCS CONCEPTS

• **Information systems** → **Data compression**; *Data encoding and canonicalization*; • **Hardware** → *Energy metering*.

KEYWORDS

Energy dataset, data compression, smart meter, integer coding

ACM Reference Format:
John R. Ward and Sean K. Barker. 2020. Powerstrip: High-Performance Compression for Energy Data. In *The Eleventh ACM International Conference on Future Energy Systems (e-Energy'20), June 22–26, 2020, Virtual Event, Australia.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3396851.3397716

1 INTRODUCTION

Modern smart buildings are dependent on high-quality data collection describing the building, occupants, activity patterns, etc. Interest in sustainable smart buildings has been driven in large part by the widespread availability of energy data, traditionally gathered from household smart meters. From 2011 to 2016, American residential smart meter coverage rose from about a quarter of all homes to almost half, for a total of 69 million installations [43].

Traditional smart meters collect electrical usage data at the aggregate level only—i.e., recording the consumption of an entire building. However, such data is inherently limited; many optimizations require device-level data, such as knowing the fine-grained consumption of air conditioners or a dryer in a home to determine optimized device schedules or flatten peak energy consumption [4]. Furthermore, field studies have shown significant energy consumption reductions through customers' knowing their own detailed energy patterns [3]. The importance of device-level data has spawned the active research area of non-intrusive load monitoring (NILM), which seeks to break down aggregate readings into individual devices [19]. However, accurate NILM in real-world environments (e.g., with hundreds of devices) remains largely unsolved.

As a more scalable solution, modern energy meters increasingly offer submetering capabilities and can directly record circuit or device-level readings. Consumer-grade meters such as eGauge [14] or Brultech [9] can be installed in an existing building for a few hundred dollars and provide data from each individual circuit. Smart outlets and device-level meters provide even finer-grained data and can be purchased for around $20 USD, providing visibility into specific devices on multi-device circuits.

While finer-grained energy data presents many opportunities for energy optimization and analysis, a resulting challenge to address is the corresponding explosion in data to be stored. Relative to the historical standard of building-level data, the data increase in a typical home might be 2 orders of magnitude (roughly 100 individual devices, ranging from small devices to larger appliances) or even more in office buildings. The increasing time granularity of meter data over time further compounds this data collection challenge. For example, assuming a 1 Hz data resolution and a conservative 4 bytes per power reading (and ignoring all likely metadata, e.g., timestamps) a home with 100 devices would generate roughly 33 MB of data per day. While reasonable for a small number of homes, this rate of data collection becomes much more challenging when conducted at a city or utility scale – e.g., a utility with 100,000 customers would generate roughly 100 TB of data each month. Distribution of such data to users is particularly problematic, but has not received significant attention owing to the typical sizes of existing public device-level energy datasets, which rarely span more than around ten buildings [12]. Going forward, however, we expect larger-scale datasets in the style of Dataport [37] to become increasingly popular for conducting city-scale data analyses.

The natural approach to addressing this problem is data compression, a widely-studied area with decades of historical work. Real-world energy data exhibits a number of useful properties that can be effectively exploited for compression, such as long periods of inactivity and gradual energy changes over time. However, while a multitude of compression algorithms exist that may be applied to energy data, we are aware of no efforts to compress energy data that specifically target these properties.

[*]Work performed while a student at Bowdoin College.

Contributions. In this paper, we present Powerstrip, a fast, highly-compressive, and nearly-lossless compression algorithm for integer energy data. Our primary contributions are threefold:

1. Analysis. Using a real-world dataset, we highlight a number of useful properties of energy data that can be exploited to efficiently compress such data. These properties (such as inactivity periods and small power steps) are especially prevalent in the device-level data for which Powerstrip is optimized.

2. Compression. We present the Powerstrip integer compression algorithm, which leverages the properties we identify to achieve aggressive compression rates with low overhead on large-scale datasets. The algorithm is fast, nearly lossless, and regularly achieves compression rates over 90% on many kinds of power data.

3. Evaluation. We evaluate Powerstrip against a wide variety of state-of-the art compression algorithms, including time series compressors, dictionary compressors, and integer coders. Experiments and case studies using multiple real-world datasets demonstrate that Powerstrip regularly achieves the best speeds and compression rates of the algorithms tested by significant margins.

We first discuss notable existing compression methods in Section 2 that can be applied to energy data. We then present a study of real-world smart meter data in Section 3 that motivates our compression design. The Powerstrip compression algorithm itself is presented in Section 4 and our implementation summarized in Section 5. We evaluate Powerstrip against the reference algorithms in Section 6 and finally conclude in Section 7.

2 RELATED WORK

Energy data normally consists either of low-frequency power measurements (typically 1 Hz or less) or high-frequency waveform measurements (typically 5 kHz or more). While high-frequency data is especially suited to compression [26], collecting such data generally requires specialized equipment [18, 27]. As such, we focus on lower-frequency readings, and outline several well-known techniques that may be used to compress such data below.

2.1 Quantization and Downsampling

Simple lossy approaches trade off data fidelity and storage size by compressing either the power resolution of the data (quantization) or the time resolution of the data (downsampling). Quantization maps a large range of values (e.g., the continuous interval $[0, 100]$) to a discrete set S (e.g., $\{5, 15, \ldots, 95\}$), which can then be coded using $\lceil log_2(|S|)\rceil$ symbols. The number and size of the bins trade off compression and fidelity to the original data. Uniform quantization partitions the range of the signal into n uniform bins, while Gaussian quantization uses thinner bins near the mean and wider bins at the tails of the distribution. This approach has been used to compress 1 Hz smart meter data by over 99% while maintaining accuracy in a random forest classifier [45].

Downsampling methods include simple decimation (pick every nth value) and Piecewise Aggregate Approximation [24], in which the time series is represented as a series of constant functions placed at the average of the data aggregated underneath. Piecewise regression extends this technique to higher degree polynomials [15]. Clustering algorithms are evaluated on power data in [30] and [17] for varying time resolutions and find strong performance on 8-minute resolution that degrades significantly at 1-hour resolution. A notable limitation of quantization and downsampling methods is a tendency to miss outliers (which often signal events of interest).

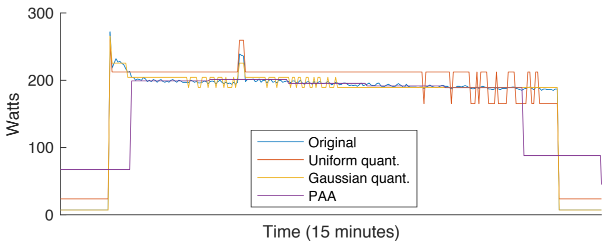

Figure 1: Refrigerator load data compressed using quantization or downsampling.

A visual comparison of quantization and downsampling methods is shown in Figure 1, which depicts a 1 Hz refrigerator load signal compressed using uniform quantization (using 50 bins), Gaussian quantization (using 50 bins), and PAA downsampling (using functions placed at intervals of 40).

2.2 Transform Coding

Signal processing has produced many other techniques that may be used to compress energy data. Popular early approaches included various linear transforms [33], which project a given signal onto some basis in a vector space, yielding coefficients of the basis vectors that can be coded. These transforms aim to code a signal $\mathbf{x}$ as $\mathbf{x} = \mathbf{D}\gamma$, for coefficients γ of rows of a dictionary $\mathbf{D}$. Most transform methods on power load signals use either the Discrete Wavelet Transform or Wavelet Packet Transform [38], which decompose the signal into a basis of short wave bursts known as wavelets. These methods are effective at high resolutions (e.g., thousands of Hz) but less so at the lower resolutions (e.g. 1 Hz) typical of most power meters. At lower resolutions, oscillations and periodic features will have a period less than the sampling resolution, and so sample points will capture very little of the oscillation's actual structure. For example, the Daubechies wavelet transform is applied to load data with samples every 15 to 60 minutes in [47] and achieves high compression but with significant error. Other challenges encountered by these methods include the difficulty of representing both spikes and periodic activity using mathematically-convenient bases of only impulses (spikes) or sinusoids (oscillations) [39].

2.3 Dictionary Coding

In dictionary coding, a basis larger than the signal dimensionality is used (i.e., an overcomplete basis) in order to approximate the original signal. In this context, the core problem is *sparse approximation*, in which n scaled basis vectors (or "atoms") are chosen from the overcomplete basis (dictionary) such that the difference between the sum of the scaled atoms and the signal being approximated is minimized. Formally, given a signal f, a target of n atoms from a dictionary D, and $||\gamma||_0$ non-zero entries in γ, the goal is to compute:

$$\min_{\gamma} ||f - D\gamma||_2^2 \quad \text{subject to} \quad ||\gamma||_0 \leq n$$

This problem is NP-hard, so various efficient approximations exist. Matching Pursuit [29] is a simple greedy algorithm: for every iteration, simply choose the atom that best matches the difference between the signal and the current approximation. More recent methods include Orthogonal Matching Pursuit [13], which updates all coefficients on every iteration, and Basis Pursuit [10], which converts the optimization into a linear programming problem.

A key challenge in dictionary coding is building dictionaries that effectively approximate the signal. A well-known dictionary learning algorithm is K-SVD [1], which alternates between coding and dictionary improvement using k-means clustering and Orthogonal Matching Pursuit. K-SVD has been used in [44] to approximate aggregate load data by learning a dictionary of partial usage patterns.

Lossless data compression methods like LZ77, LZ78, and LZMA are also considered "dictionary coders," though they share little in common with Matching Pursuit or K-SVD. In these methods, symbols are encoded as references to past instances of the same symbol through a sliding window history or other dictionary model. However, since power data rarely has exact repeats due to noise [8], exact dictionary coders are less efficient for power data than for other data. These methods have achieved high compression ratios on very high resolution energy data (e.g., 10 kHz) [22, 40], but have shown poorer performance on lower resolutions [41].

2.4 Integer Encodings

Integer coders can be used to compactly represent integer data sequences, such as (integer-valued) energy data time series. A classic technique is a variable-byte ("varint") encoding [46], which encodes integers in blocks of 8 bits, using the lower 7 bits to store data of the actual integer and the top bit to indicate whether another block for the integer follows. In addition to simplicity, a significant benefit of varint encoding is typically high performance due to the byte alignment of the data [35].

Examples of more advanced integer coders include Simple8b [2] and FastPFOR [28]. Simple8b codes values in blocks of 64 bits, with the first 4 bits denoting the number of bits used for each integer packed into the other 60 bits. FastPFOR uses a similar approach but adds special handling for outliers and SIMD vectorization for performance gains.

3 LOAD DATA PROPERTIES

Real-world power load data exhibits many useful characteristics that can be exploited for effective compression. Here, we highlight several such properties that motivate the design of Powerstrip by conducting a data analysis on real-world load data from the well-known REDD dataset [25]. This dataset consists of consumption data from over 100 circuits across six residential homes (including dedicated circuits on many typical appliances, e.g., refrigerator, dishwasher, etc), and is collected at 1 Hz aggregate resolution and roughly 1/3 Hz resolution on the individual circuits.

3.1 Frequent Inactivity

Even during periods of low activity (such as when occupants are away or sleeping), buildings consume highly variable levels of energy. For example, background energy events such as refrigerator compressor cycles occur regularly even during relatively dormant periods, resulting in a variable pattern of energy consumption at the building level.

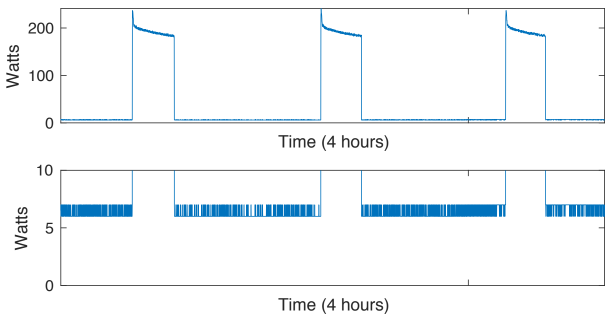

Figure 2: Power consumption of a refrigerator (top) showing oscillations during inactivity (bottom, zoom-in).

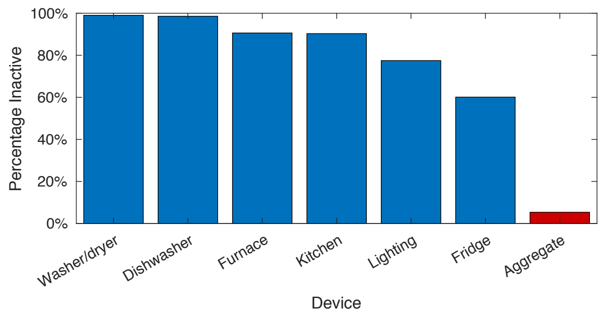

Figure 3: Most individual devices spend the majority of their time in an inactive state.

At the device level, however, most devices exhibit frequent, lengthy periods of inactivity—either when switched off completely (such as a toaster) or when simply not in use (such as an idle computer). During such periods, energy consumption is largely flat (though not necessarily zero). Inactivity may not produce constant power readings, however, due to noise, meter error, averaged measurement intervals, etc. For example, Figure 2 shows the consumption of a refrigerator (top) along with a zoom-in of inactive periods (bottom). While the inactive periods are trivial to recognize visually, readings during these periods slightly but continuously oscillate.

As a simple way to measure such periods, we consider the following method: choose the mode of the data (i.e., the most common consumption level), and label all readings below $(mode + \epsilon)$ watts as "inactive" for some reasonably small ϵ. This approach is likely to flag most periods of inactivity, whether they manifest as near-zero consumption or otherwise. Using $\epsilon = 5$, Figure 3 shows the proportion of time spent in an inactive state for a variety of device types in the REDD dataset, averaged across all devices of that type. Most devices are considered inactive for well over 50% of the data, and many exceed 90%; several prominent large appliances (such as dishwashers and washers/dryers) show inactivity of over 99%. Even a typical "always-on" device such as a refrigerator shows (on average) inactivity of 66%. In contrast, the average aggregate signal in the data is inactive for less than 10% of the time. This result reflects that while individual devices are largely dormant, the aggregation of many such devices is highly active.

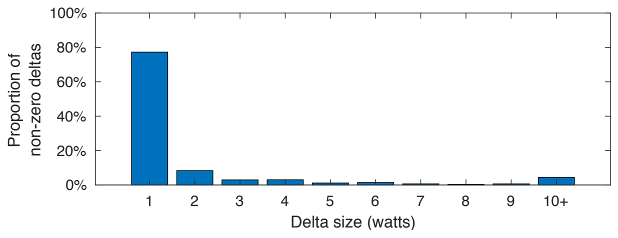

Figure 4: Delta distribution for the complete REDD dataset.

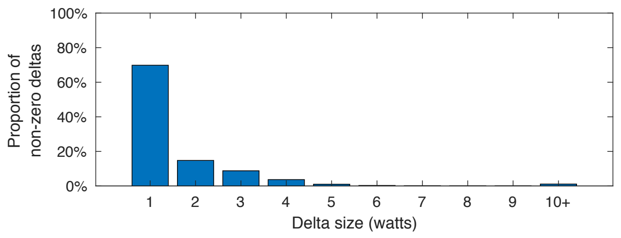

Figure 5: Delta distribution for a single refrigerator.

3.2 Small Deltas

In most cases, the datapoints of most interest in energy traces are the largest steps, as these deltas typically signify events of interest. For instance, a 60W positive delta for a light likely indicates that the bulb was turned on, while a 100W positive delta for a refrigerator likely indicates the start of a compressor cycle. However, large steps such as these are infrequent in real data when compared to smaller steps of only a few watts. Figure 4 shows the proportion of non-zero deltas in the REDD dataset for a variety of delta sizes. The resulting distribution shows a typical "long tail" shape: 1W deltas are the most prevalent by far (nearly 80% of all non-zero deltas), while less than 10% of non-zero deltas are 10W or greater.

This distribution is true of most individual devices as well. For example, Figure 5 shows the delta distribution for the refrigerator previously pictured in Figure 2. For this device, the middle 98% of deltas fall within the range $[-6, 5]$, even though the full set of deltas (including most events of interest, such as compressor activations) spans the much larger range $[-200, 250]$.

3.3 Imperfect Repetitions

Given the number of periodic devices in buildings (refrigerators, ACs, etc.), typical energy data contains frequent repetitions of device behavior. For example, the refrigerator in Figure 2 appears as the same basic usage pattern repeated at a mostly regular interval.

To investigate the actual degree of data repetition in this device, we analyzed the number of repetitions of ten-symbol words in the data (i.e., ten particular readings occurring consecutively). Table 1 shows the proportion of words with occurrences falling under several illustrative thresholds. We see that a significant majority of ten-symbol words are very uncommon in the data—e.g., over half of the words in the data (56.2%) occur less than 0.1% of the time within the data. In fact, the only frequently occurring pattern in the data is the constant word $k, k, \ldots, k$ (primarily during inactive periods). In short, the number of exact data repetitions is quite low, even in a highly regular device such as a refrigerator.

Occurrence	Proportion of words
< 0.01%	28.1%
< 0.1%	56.2%
< 1%	82.2%
< 10%	87.6%

Table 1: Most ten-symbol words in data from a refrigerator occur less than 0.1% of the time.

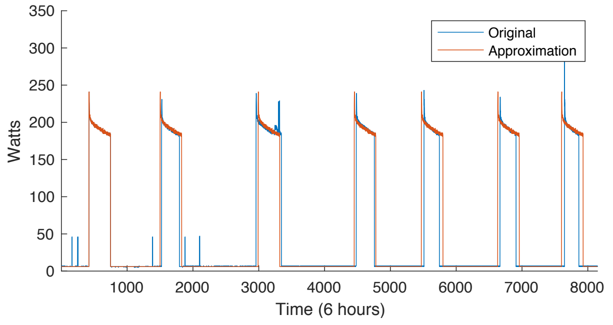

Figure 6: An approximation of a refrigerator constructed using a single compressor cycle.

A natural alternative to looking for exact repetitions is to employ approximations. As a case study of this approach, we approximated the refrigerator data by replacing each compressor cycle over a 6-hour period with copies of a single (representative) compressor cycle, as depicted in Figure 6. We then measured the resulting root mean square error (RMSE) to evaluate the accuracy of the approximation. While the approximation appears visually accurate, a nontrivial amount of error is introduced relative to the original data. In fact, the error introduced by regularizing the compressor cycles (RMSE=45.1) is 75x larger than by flattening out all oscillations between compressor cycles (RMSE=0.6). This result suggests that approximations may be more cheaply applied to inactive periods than to active periods. This insight influences the compression design of Powerstrip, as described in Section 4.

4 COMPRESSION DESIGN

Here, we present the Powerstrip compression algorithm for integer energy data. Powerstrip is designed to exploit empirical properties of real-world energy data (detailed in Section 3) and is particularly suited to compressing device-level data. Although a small degree of lossiness is leveraged to improve compression, the degree of this loss is near-zero in practice. Furthermore, Powerstrip is extremely lightweight, achieving compression and decompression speeds faster than both general-purpose solutions (e.g., `gzip`) and more specialized compression algorithms.

A data sequence compressed by Powerstrip consists of a series of *blocks*, each encoding a contiguous chunk of the original data. In the expected case of multiple devices, each device is represented by a separate data sequence and compressed independently. Powerstrip itself does not encode timestamps or other index values (as in time series databases and other related work [7]). As such, reconstructing a time series of *(time, watts)* pairs requires either a consistent recording interval and two pieces of metadata (start time and interval size) or external storage of index values.

For a given data sequence input, Powerstrip begins by dividing the uncompressed input data into fixed-sized blocks (by default, 128 KB each) and then compresses each block independently. Compression of each block operates in three phases:

(1) Detection and flattening of inactive intervals.
(2) Delta and zigzag encoding.
(3) Bit-packing and Huffman coding.

The output of the final phase is a compressed block of variable size (up to the original block size). Each of the three compression phases are detailed below.

4.1 Inactivity Flattening

The goal of the first phase is to identify and flatten periods of inactivity in the block, such that they need not be stored at all. As shown in Section 3.1, real-world devices exhibit a high proportion of such periods, even for "always-on" background devices.

A basic approach is to use run-length encoding (i.e., storing start times and durations) to compress consecutive duplicate readings. However, as observed previously in Figure 2, periods of inactivity often manifest as irregular oscillations within a small range rather than duplicate readings, largely defeating run-length encoding. Instead, we use a slight variant of the approach introduced in Section 3.1 to estimate periods of inactivity. Given the mode of the data within the block (which we term the *blockmode*) and global parameter ϵ, we label all data within the range $[blockmode - \epsilon, blockmode + \epsilon]$ as inactive, and replace all such data with the blockmode itself. In doing so, all such datapoints can be eliminated, as the blockmode simply becomes the "default" value within the block. Note that this default value may vary for different devices, as well as for different blocks of the same device. The setting of ϵ determines the degree of flattening, and trades off between compression and lossiness. In practice, the default setting of $\epsilon = 3$ is sufficient to capture most gains without introducing significant lossiness (demonstrated in Section 6).

Effective flattening assumes that the block actually contains periods of inactivity. Some highly active blocks may not contain such periods, however, in which case flattening may incur data loss without significant compression benefit. To guard against this case, we only apply flattening if the blockmode represents at least 10% of the data readings. To justify this choice, Figure 7 shows the frequency of the mode across all devices of several representative types in the REDD dataset (broken down by quartiles), along with the aggregated data. We see that the only cases where the mode occurs less than 10% of the time are aggregate readings and a small number of devices with unusual characteristics (e.g., a highly variable floor – though the proportion within individual blocks would likely be higher). Thus, we are able to apply the flattening step in the majority of cases while still protecting against pathological cases where unnecessary loss might be introduced.

We note that the inactivity flattening step is the only (slightly) lossy step in the complete algorithm. If ϵ is set to 0, then Powerstrip operates as a fully lossless compression algorithm.

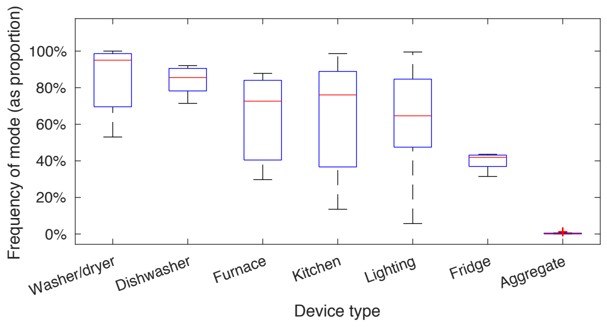

Figure 7: Most devices have a frequently occurring mode.

4.2 Delta Encoding

Once inactive periods are removed via flattening, the remaining active periods (which we term *segments*) are extracted as (*index, length*) pairs marking their positions within the original data. Note that these values do not convey time in any way, but are merely relative to the original integer sequence (which may or may not consist of fully regular readings).

We apply two techniques to compress the data within each segment. We first apply delta coding, which replaces each value by the difference from the preceding value. For example, the sequence $\{3, 5, 5, 6, 2, 0\}$ would be delta-coded to $\{3, 2, 0, 1, -4, -2\}$. We then apply variable-length zigzag encoding [16] to the resulting deltas. Intuitively, this encoding orders the signed integers as $\{0, -1, 1, -2, 2, \ldots\}$ and then uses the minimum number of bits necessary to encode the values (e.g., only 2 bits would be needed to encode values in the set $\{0, -1, 1\}$). For k-bit source integers and following delta coding, encoding and decoding is performed using standard C bitwise operators as follows:

$$\text{encode}(x) = (x << 1) \wedge (x >> k - 1)$$
$$\text{decode}(x) = (x >> 1) \wedge -(x \;\&\; 1)$$

Following the analysis in Section 3, we expect most deltas to be small and therefore fit within the zigzag encoding using only a few bits. However, the small number of larger deltas typical in power data may explode the number of bits required to represent the entire set. We handle this problem using an outlier array, similar to [48]. Specifically, we fix m bits to be used in zigzag encoding and then designate all values outside the representable $[-2^{m-1}, 2^{m-1}]$ range as outliers. Outliers are then stored in a separate array as $(k+1)$-bit deltas (the extra bit is needed to move from the largest values to the smallest or vice versa). The special value consisting of all m 1's is used as an outlier marker, indicating that the corresponding delta is stored in the outlier array rather than zigzag encoded.

The value of m is chosen on a per-block basis in order to minimize the final compressed size. Specifically, if $f(m)$ is the fraction of the signal that can fit in m bits, we solve the following for k-bit inputs:

$$\min_{m \in [1,k]} \; m \cdot f(m) + (k+1) \cdot (1 - f(m))$$

Given a small value of k, solving this optimization problem is easily done by building a histogram during a single pass over the data. The default setting of k is 16, which allows for signed data values up to 32767 (positive or negative).

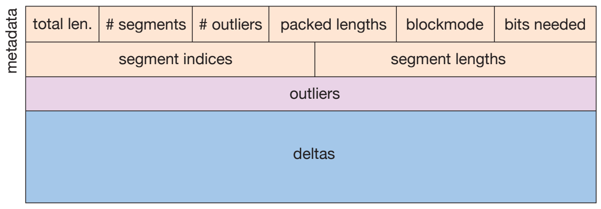

Figure 8: The data layout of a single compressed block.

4.3 Bit-packing and Huffman coding

Following delta and zigzag encoding, each block is stored in a 3-part format, as shown in Figure 8:

(1) Metadata – the blockmode, segment information, bits used for zigzag encoding, etc.
(2) Outliers – the outlier array from the encoding phase.
(3) Deltas – the encoded deltas (normally the largest section).

Two additional techniques are then applied to compress the block further. First, the outlier and delta sections are bit-packed using off-the-shelf methods from SIMDComp [28], a fast vectorized bit-packer that makes use of SSE SIMD instructions to improve performance. Second, the entire block is compressed using Huffman coding [20] (a standard form of entropy encoding). The order of these steps could be reversed, but Huffman coding is relatively expensive (and thus better applied to smaller bit-packed data), and Huffman-coded data does not benefit significantly from subsequent bit-packing. Both of these factors have been previously noted in [7].

The final output for a given device consists of all its compressed blocks laid out sequentially within a single file.

4.4 Decompression

Decompression in Powerstrip follows intuitively from the compression design. Specifically, decompressing each block requires only a few low-level operations and a bit-unpacking call, as follows: (1) delta unpacking; (2) zigzag decoding; (3) setting the blockmode via a single `memset` call; and (4) copying active segments into place.

4.5 Limitations

The primary limitation of Powerstrip is a natural consequence of its design as an integer coder, which makes it unsuitable for data that requires floating-point precision. Such data most notably includes high-frequency measurements of current and voltage waveforms (generally taken at 5 kHz or greater). However, our focus is on data collected from typical utility and consumer meters, which generally cannot record high-frequency waveform data [18, 27].

Additionally, the compression effectiveness achieved by Powerstrip is largely dependent on the expected data characteristics described in Section 3. These characteristics are less prevalent in aggregate data than in device-level data, limiting Powerstrip's advantages when applied to aggregate data. However, the volume of device-level data (when such data is available) is typically much greater than of aggregate data. Thus, Powerstrip is most effective in the cases where compression is most important.

5 IMPLEMENTATION

Powerstrip is implemented in C++ as an open-source, publicly available[1] command-line compression and decompression library, with bit-packing provided by the SIMDComp library [36]. The command-line utilities convert between the binary, compressed data layout of Powerstrip and a decompressed, 16-bit binary format. For a typical dataset consisting of multiple device traces, each trace is stored in a separate and fully independent file.

Powerstrip neither requires nor stores any timestamp information in the data files. In the simplest case of a perfectly regular time series, a user wishing to distribute the time series would provide the compressed data file along with the timestamp of the first reading and the measurement interval. For data that is not fully regular or contains missing datapoints, several approaches are possible. If there are few missing readings, two reasonable approaches are either filling in missing readings with the previous measurement or storing the timestamps of missing datapoints outside of the main compressed file. For more irregular data, explicit timestamps may need to be stored alongside the compressed power readings. In this case, timestamps can be compressed using double-delta coding (i.e., assuming a default inter-data interval and storing deltas from this interval), which will replace consistently-spaced intervals with 0's.

While our current implementation operates on a single machine, a real-world deployment of Powerstrip would likely be in a server or data center environment on behalf of a utility. In such a scenario, massive amounts of meter data would be collected and stored for later analysis (e.g., user analytics, evaluation of NILM algorithms, demand response techniques, etc). A utility or other data aggregator might also wish to anonymize the data to maintain privacy; while such issues are orthogonal to the compression goals of Powerstrip, privacy-preserving techniques [21, 32] could be applied prior to compression to provide privacy guarantees.

6 EVALUATION

We evaluate the compression, accuracy, and efficiency of Powerstrip using real-world data from the REDD dataset [25] and compare to the 10 reference algorithms listed in Table 2. The reference algorithms include state-of-the-art compression algorithms spanning most well-known approaches (integer coders, time series coders, dictionary coders, etc). For Ztsd, we consider three configurations: fastest (Zstd-1), best compression (Zstd-22), and an intermediate level (Zstd-5). For Sprintz, we use the most compressive variant (FIRE forecasting and Huffman coding). Quantization and PAA (which lack reference implementations) are implemented in Matlab. For all other algorithms, we use highly-tuned, off-the-shelf implementations integrated into the `lzbench` benchmarking tool.

As lossy algorithms are normally tunable between compression and lossiness, we consider lossless and lossy algorithms separately. We evaluate Powerstrip along three general metrics:

(1) **Compression** – the storage savings achieved by Powerstrip versus the lossless reference algorithms.
(2) **Lossiness** – the compression vs. lossiness tradeoff achieved by Powerstrip versus the lossy reference algorithms.
(3) **Speed** – the compression and decompression speeds achieved by Powerstrip versus the fastest reference algorithms.

[1]https://github.com/joliv/powerstrip

Name	Type	Description
libdeflate [6]	Lossless	LZ77 dictionary coder, as in `gzip`
LZMA [31]	Lossless	Extension of LZ77
Zstd [11]	Lossless	LZ77+asymmetric numeral systems
Simple8b [2]	Lossless	Integer coder
FastPFOR [28]	Lossless	Optimized integer coder
Sprintz [7]	Lossless	IoT integer time series compressor
Uniform quant.	Lossy	Quantization
Gaussian quant.	Lossy	Quantization
PAA	Lossy	Downsampling
K-SVD [34]	Lossy	Sparse dictionary learning

Table 2: The reference compression algorithms considered.

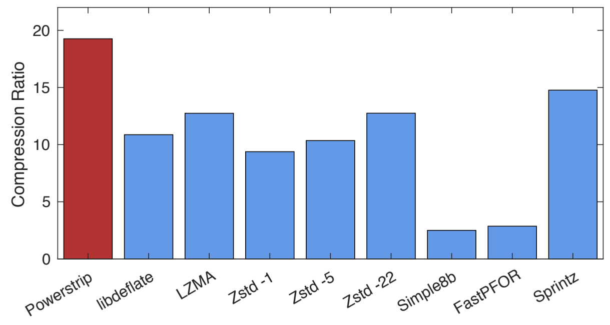

Figure 9: Powerstrip achieves superior compression ratios to the lossless algorithms on the entire REDD dataset.

Test data is drawn from REDD unless otherwise noted. As a preprocessing step, we convert the original CSV text data into a binary format of signed 16-bit integers and eliminate all additional metadata (e.g., timestamps).

6.1 Compression

We first consider compression effectiveness in Powerstrip by measuring the compression ratio of the entire REDD dataset (including an "aggregate" circuit per home), computed as the uncompressed size divided by the compressed size (larger is better). The compression ratios attained by Powerstrip and the 6 reference lossless algorithms (plus the two additional Zstd variants) are shown in Figure 9. We see that Powerstrip achieves the greatest compression (a 94.8% size reduction), resulting in a compressed size 23% smaller than that of the best reference algorithm (Sprintz).

We next consider compression of individual circuits in the dataset. For each algorithm, Figure 10 shows the distribution of compressed sizes across all circuits, with aggregated circuit results plotted separately. To account for size variations across devices, results are plotted relative to Simple8b as a baseline. Outlier circuits are plotted separately to demonstrate the "worst-case" devices for compression. We see that Powerstrip consistently outperforms all of the reference algorithms when compressing individual devices (i.e., non-aggregated readings). For these devices, Powerstrip also demonstrates more consistent compression than the other algorithms, with a spread between the mean and worst outlier of only 25%. For aggregated circuits, Powerstrip is competitive but not significantly superior to the best reference algorithms. The likely reason for this difference is the deactivation of inactivity removal on many of the aggregated circuits (degrading compression but making Powerstrip fully lossless in these cases). Overall, Powerstrip demonstrates best-in-class compression performance across all devices and significantly outperforms the reference algorithms on individual circuits.

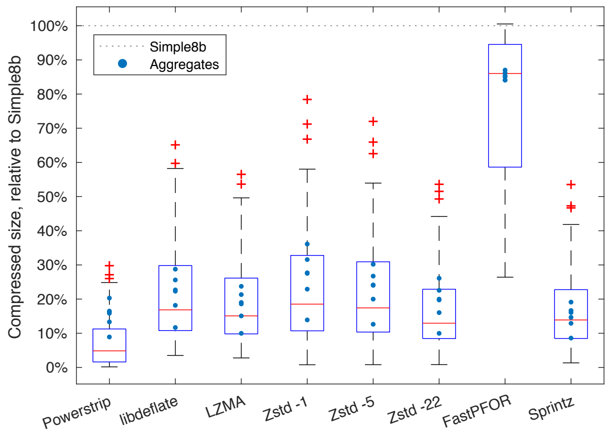

Figure 10: Compressed size distributions across circuits for Powerstrip and reference lossless algorithms.

6.2 Lossiness

The lossy reference algorithms we consider are tunable between compression and data fidelity. As such, we first conduct an experiment in which we compress a refrigerator trace using a variety of parameter settings for each reference algorithm. For K-SVD, we fix 4 atoms and 10 training iterations, then vary the size of the dictionary from 10 to 1000. For uniform and Gaussian quantization, we vary the number of bins from 10 to 1000. Finally, for PAA downsampling, we vary the timestep from 1 to 100. For each compression run, we measure the compressed size and the root mean square error (RMSE), calculated for a signal x and compressed signal $\hat{x}$ of length n as the following:

$$RMSE = \sqrt{\sum_{i=1}^{n}(\hat{x}_i - x_i)^2}$$

The compressed sizes and RMSEs for all runs are shown in Figure 11, demonstrating the tradeoff curves between compression and error. The lossy algorithms display a wide variation of behavior—e.g., K-SVD generally achieves high accuracy but poor compression, while PAA downsampling generally achieves high compression but poor accuracy. The single run of Powerstrip, however, exhibits the least error of any test run, while achieving better compression than all other methods except for aggressive downsampling (which incurs high error). In short, Powerstrip achieves much greater compression at substantially lower cost than the other methods tested.

Next, we consider multiple devices using a set of fixed parameters—a 100-atom dictionary for K-SVD, 100 bins for quantization, and a PAA timestep of 100. We then compress all devices within several sets of representative devices (e.g., all dishwashers).

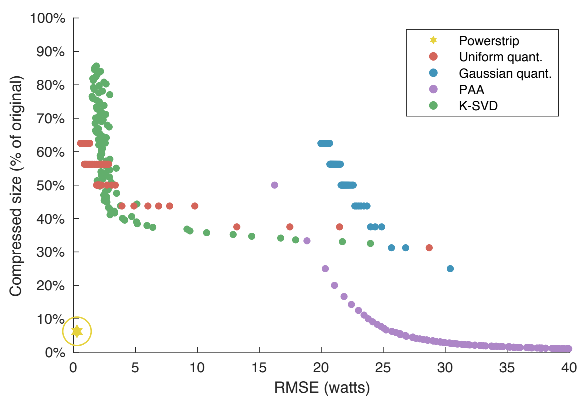

Figure 11: Powerstrip achieves a superior compression/error tradeoff to the other lossy algorithms considered when compressing a refrigerator power trace.

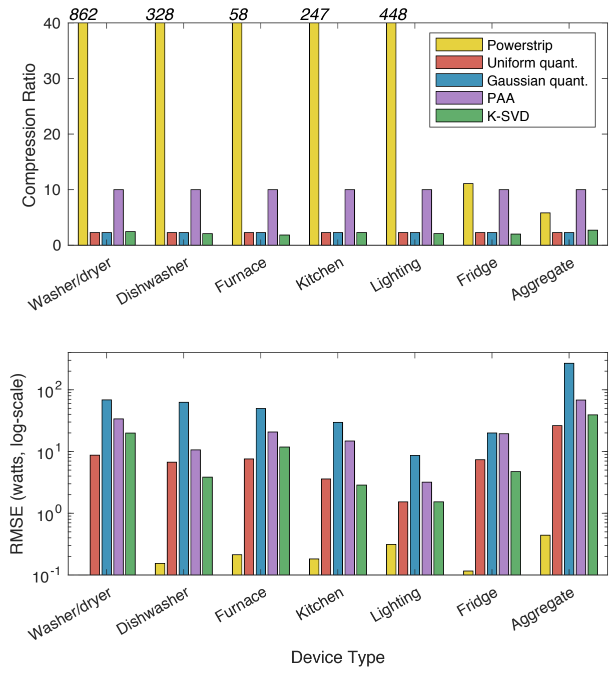

Figure 12: Average compression ratios (top) and error (bottom) of lossy methods on several device classes.

Figure 12 shows the average compression ratio (top) and RMSE (bottom, log-scale) for each device set. We see that Powerstrip achieves the best average compression ratio in nearly all cases (sometimes by multiple orders of magnitude), though aggregate data is a notable (but expected) outlier. More important, however, is that Powerstrip achieves these compression results with an average RMSE of less than 0.2 watts. This level of error is likely less than the error of the meter itself and is unlikely to be significant for any application already operating on low-frequency power data (which is inherently lossy over the measurement interval even if perfectly compressed). We also note that the design of Powerstrip limits instantaneous errors to a few watts or less (so the error distribution over time should be fairly uniform) and does not introduce any error at all during active periods.

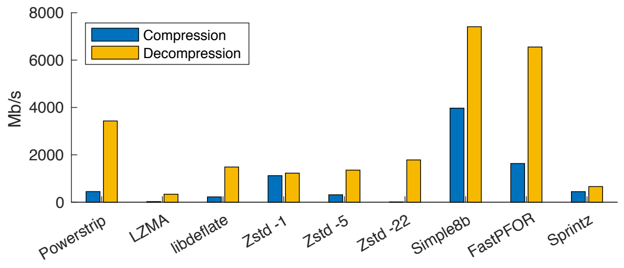

Figure 13: Average compression and decompression rates of Powerstrip and reference lossless algorithms.

6.3 Speed

We next measure compression and decompression rates across the entire REDD dataset on a 2.7 GHz Intel i5 processor with no parallelization. We report speed results for Powerstrip and the 8 lossless algorithms – as mentioned previously, the lossy approaches are omitted due to a lack of competitive implementations.

Figure 13 shows the average compression and decompression speeds across the entire dataset. The best overall performance is achieved by the integer coders Simple8b and FastPFOR – however, these algorithms achieve much less compression (see Figure 9). Among algorithms with comparable compression results, Powerstrip displays best-in-class decompression speed (3430 MB/sec – over 5x faster than Sprintz) and competitive compression speed (448 MB/sec – comparable to Sprintz and over 80x faster than Ztsd-22).

In practice, the compression speeds achieved by Powerstrip are orders of magnitude larger than needed for on-the-fly compression in individual buildings. As such, it is entirely feasible to perform on-site compression using a low-power device (e.g., a smart home hub) prior to sending data to centralized or cloud storage.

6.4 Inactivity Flattening Window Size

The range of values flattened around each blockmode is controlled by the setting of ϵ. The flattening window should be large enough to flatten most periods of inactivity but small enough to effectively constrain lossiness. To investigate its impact on compression, we varied ϵ from its default value ($\epsilon = 3$) and recompressed the complete REDD dataset. Figure 14 shows the resulting compression ratios for ϵ ranging from 0 to 10. Note that $\epsilon = 0$ represents a lossless configuration in which no flattening occurs (although the blockmode itself is still effectively removed). The most significant jump in compression occurs when moving from $\epsilon = 0$ to $\epsilon = 1$, reflecting the large number of 1W oscillations that occur during inactive periods. Modest but diminishing gains occur as ϵ is further increased (e.g., minimal incremental benefit beyond $\epsilon = 5$).

Powerstrip's standard configuration of $\epsilon = 3$ (used for all experiments unless otherwise noted) is a relatively conservative setting, which captures most gains while limiting lossiness to steps of 6W or less around the blockmode (most of which are transient steps of only a few watts). However, more aggressive compression may be

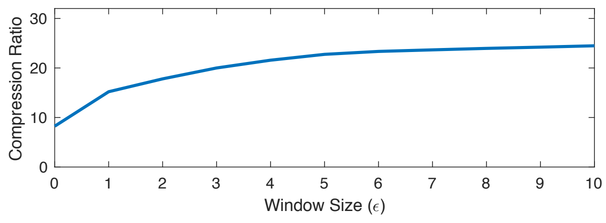

Figure 14: Powerstrip compression ratios on REDD dataset for a variety of flattening windows (ϵ values).

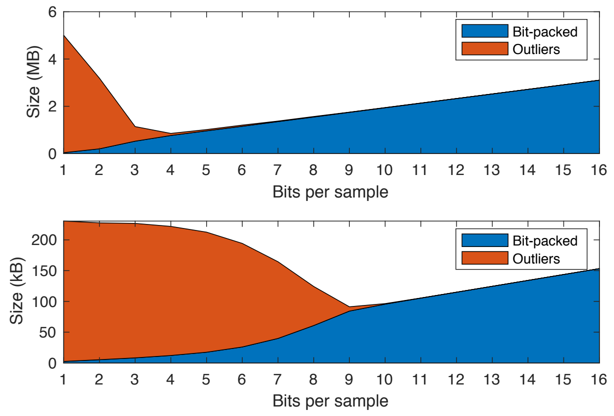

Figure 15: Compressed size of a refrigerator (top) and washer/dryer (bottom) for fixed m (bits per sample).

achieved by modestly increasing ϵ. We note that even $\epsilon = 1$ (which limits possible errors to steps of at most 2W) achieves greater compression than Zstd-22 and Sprintz on the full dataset.

6.5 Impact of Bit Selection

We also investigate the impact of m (the number of bits used in delta encoding), which affects both the number of outliers and the total data size. Smaller values of m result in more outliers but a smaller chunk of bit-packed data, while larger values of m result in fewer outliers but a larger chunk of bit-packed data. Powerstrip optimizes m in order to efficiently trade off between these approaches.

Figure 15 shows the compressed size of two device traces as m is varied: a refrigerator (top) and a washer/dryer (bottom). Here, we fix m across all blocks and measure the total compressed size of each device, broken down between outliers and deltas (metadata contributes an insignificant amount to the total). We see that at the smallest values of m, the data is almost entirely outliers, while at higher values of m, the data contains almost no outliers. More importantly, however, we see that the optimal value varies substantially across devices. In the case of the refrigerator, the optimal value is $m = 4$, while for the washer/dryer, the optimal value is $m = 9$ (and choosing $m = 4$ for this device results in roughly doubling the compressed size). This result demonstrates the importance of Powerstrip's optimization of m and shows that a static value may perform poorly across multiple devices. Note that Powerstrip's optimization of m is more flexible than illustrated here, since we optimize m on a per-block basis rather than a per-device basis.

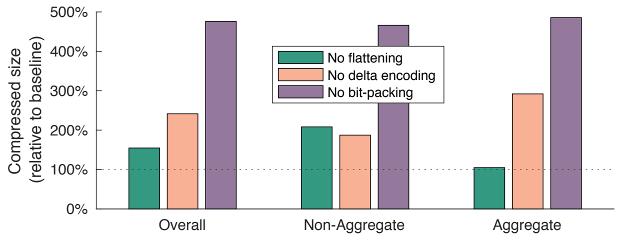

Figure 16: Compression ratios on REDD when one of the compression phases is disabled.

6.6 Compression Phases

To investigate the impact of each of the three compression phases used in Powerstrip (inactivity flattening, delta encoding, and bit-packing/Huffman coding), we disable one of the three phases and then recompress the REDD dataset. For each disabled phase, Figure 16 shows the resulting compressed size relative to the full algorithm (i.e., a larger bar indicates that the disabled phase contributes more to total compression). In addition to the overall results for the complete dataset, we also show results for just the aggregate data and for just the non-aggregate devices.

We see that the largest compression gains come from the bit-packing phase, but we note that these gains are easily achieved "off-the-shelf", without any particular design specific to Powerstrip. Powerstrip's competitive advantage is instead primarily due to the first two phases – especially inactivity flattening, which contributes very little to aggregate data (as expected), but contributes roughly 100% to individual devices. We see that the delta encoding phase is beneficial to all devices but more so to aggregate data, reflecting that aggregate data is substantially more active.

Some of the gains of inactivity flattening can be captured by simply replacing datapoints with the blockmode as a data preprocessing step, followed by running any lossless algorithm (as opposed to the latter phases of Powerstrip). We tested this idea using Sprintz and Zstd-22 and found that a "Flattening+Sprintz" hybrid algorithm produced the best compression of all (exceeding Powerstrip itself by about 13%), albeit with 70% slower decompression than Powerstrip. The better compression of this configuration is likely due in part to the FIRE forecasting algorithm used in Sprintz (which is a more compressive but slower alternative to delta coding).

6.7 Case Studies: Real-World Datasets

We next consider the use of Powerstrip as a tool to aid in the distribution of real-world datasets. Here, we consider three other well-known public energy datasets in addition to REDD: UK-DALE [23] (1 Hz data only), iAWE [5], and Dataport [37]. Dataport in particular is a much larger dataset than the others: we use Dataport's curated, 1 Hz time series dataset, which consists of 6 months of circuit-level data from 50 homes. We compress each of these datasets using Powerstrip as well as the two best-performing reference algorithms (Zstd-22 and Sprintz). The resulting compression ratio of each dataset is shown in Figure 17. Powerstrip achieves the highest compression ratio on all datasets, outperforming Sprintz (the second-best performer) by margins ranging from 18% (on iAWE) to 35% (on UK-DALE, which Powerstrip compresses by 97%).

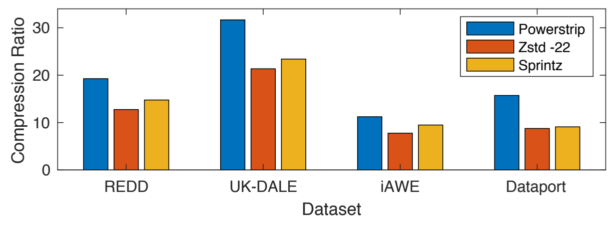

Figure 17: Compression ratios on multiple public datasets.

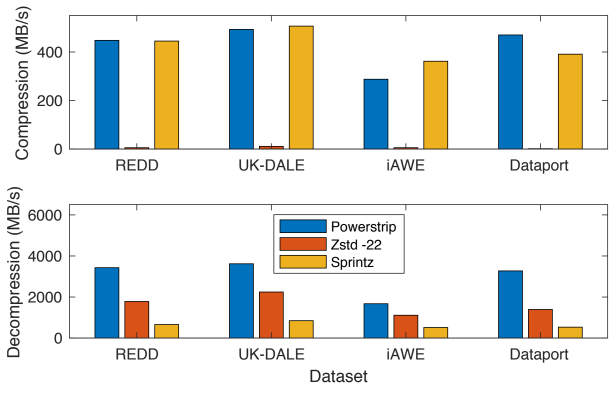

Figure 18: Compression rates (top) and decompression rates (bottom) on multiple public datasets.

Format	Size	Compress	Decompress
`gzip` (baseline)	3.9 GB	–	–
Powerstrip ($\epsilon = 3$)	1.9 GB	87s	18s
Zstd-22	3.4 GB	6.2h	30s
Sprintz	3.3 GB	80s	57s

Table 3: Compressed sizes and compression/decompression times for the compacted Dataport dataset.

Figure 18 shows compression speeds (top) and decompression speeds (bottom) for each dataset. Across all datasets, Powerstrip's overall compression rate is comparable to that of Sprintz and over 70x faster than that of Zstd-22. More importantly, decompression in Powerstrip is 4.7x faster than Sprintz and 1.8x faster than Zstd-22.

For real-world context, we consider the particular task of distributing the Dataport dataset to users. As distributed by the publishers, this dataset consists of roughly 20 GB of `gzip`-compressed text files (over 170 GB uncompressed). Much of this size is due to textual metadata (especially timestamp strings), so we first chronologically order all power readings and then discard all extraneous metadata, resulting in 34.6 GB of uncompressed data. This compacted data compresses to 3.9 GB using `gzip`, which establishes a "baseline" distribution size for the dataset.

We next consider distributing this dataset using Powerstrip, Zstd-22, or Sprintz. Table 3 shows the resulting compressed sizes, as well as the compression and decompression times required in each case on our test machine (omitted for `gzip` due to the textual data format). Powerstrip produces the smallest data archive by a significant margin using the default $\epsilon = 3$, saving over 1.5 GB compared to both Zstd-22 and Sprintz and more than halving the baseline `gzip`-compressed size. Using a more conservative $\epsilon = 1$, Powerstrip produces a data size of 2.9 GB – about 50% larger than $\epsilon = 3$ but still smaller than both Zstd-22 and Sprintz. These results demonstrate that Powerstrip is uniquely well-suited to distributing large-scale energy datasets to users.

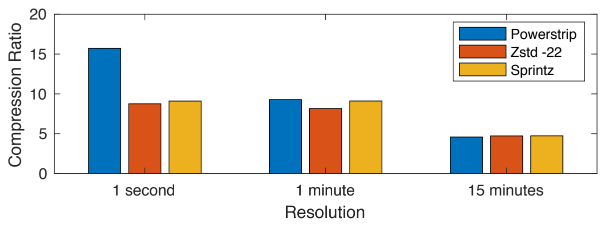

Figure 19: Compression ratios on Dataport data of multiple resolutions.

6.8 Data Resolution

We lastly investigate the impact of data resolution on Powerstrip's performance using lower-resolution data provided in Dataport alongside the primary 1-second data. Here, we use both the 1-second resolution data as well as 1-minute and 15-minute data from the same 50 houses over the same 6 month period. We also include 1-minute and 15-minute data from 23 additional houses (homes included in Dataport's curated time series dataset for which 1-second data is not currently available). Figure 19 shows the compression ratios achieved on each of the three Dataport subsets – the primary 1-second data (50 homes), the 1-minute data (73 homes), and the 15-minute data (73 homes). We see that compression generally improves when moving to higher resolutions, as has been demonstrated previously [42]. However, only Powerstrip demonstrates a significant improvement at 1-second, which likely stems from the greater degree of flattening possible at higher resolutions.

7 CONCLUSIONS

This paper presents Powerstrip, a compression algorithm for integer energy data that focuses on device-level data, high compression ratios, and low overhead. The design of Powerstrip leverages common characteristics of real-world energy data, along with a minimal degree of lossiness, in order to provide fast but effective compression. We conduct experiments on multiple real-world datasets and compare to 10 different state-of-the-art compression algorithms. Our results show that Powerstrip regularly exceeds the compression ratios of the reference algorithms (by as much as 35% compared to the best reference algorithm) while simultaneously offering the fastest decompression speeds by wide margins. We also find that Powerstrip introduces near-zero data loss in practice. Our case studies demonstrate the potential of Powerstrip for large-scale energy data storage and distribution, challenges that will only grow in importance as the quantity of energy data increases over time.

Acknowledgements. We would like to thank our shepherd and the anonymous reviewers for their thoughtful feedback and suggestions that improved this work.

REFERENCES

[1] M. Aharon, M. Elad, and A. Bruckstein. 2006. K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation. *IEEE Transactions on Signal Processing* 54, 11 (Nov. 2006), 4311–4322. https://doi.org/10.1109/TSP.2006.881199

[2] Vo Ngoc Anh and Alistair Moffat. 2010. Index Compression Using 64-Bit Words. *Softw. Pract. Exper.* 40, 2 (Feb. 2010), 131–147.

[3] Ian Ayres, Sophie Raseman, and Alice Shih. 2009. *Evidence from Two Large Field Experiments That Peer Comparison Feedback Can Reduce Residential Energy Usage.* Technical Report w15386. National Bureau of Economic Research, Cambridge, MA. https://doi.org/10.3386/w15386

[4] Sean Barker, Aditya Mishra, David Irwin, Prashant Shenoy, and Jeannie Albrecht. 2012. SmartCap: Flattening Peak Electricity Demand in Smart Homes. In *2012 IEEE International Conference on Pervasive Computing and Communications.* IEEE, Lugano, Switzerland, 67–75. https://doi.org/10.1109/PerCom.2012.6199851

[5] Nipun Batra, Manoj Gulati, Amarjeet Singh, and Mani B. Srivastava. 2013. It's Different: Insights into Home Energy Consumption in India. In *Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings - BuildSys'13.* ACM Press, Roma, Italy, 1–8. https://doi.org/10.1145/2528282.2528293

[6] Eric Biggers. 2016. Libdeflate.

[7] Davis Blalock, Samuel Madden, and John Guttag. 2018. Sprintz: Time Series Compression for the Internet of Things. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2, 3 (Sept. 2018), 1–23. https://doi.org/10.1145/3264903

[8] Davis W. Blalock and John V. Guttag. 2016. EXTRACT: Strong Examples from Weakly-Labeled Sensor Data. In *2016 IEEE 16th International Conference on Data Mining (ICDM).* IEEE, Barcelona, Spain, 799–804. https://doi.org/10.1109/ICDM.2016.0093

[9] Brultech (2019). BrulTech ECM-1240 Energy Monitor. http://www.brultech.com/products/ECM1240/. Accessed January 2020.

[10] Scott Shaobing Chen, David L. Donoho, and Michael A. Saunders. 1998. Atomic Decomposition by Basis Pursuit. *SIAM Journal on Scientific Computing* 20, 1 (Jan. 1998), 33–61. https://doi.org/10.1137/S1064827596304010

[11] Yann Collet and Chip Turner. 2016. Smaller and Faster Data Compression with Zstandard.

[12] Datasets (2019). Public Data Sets for NIALM. http://blog.oliverparson.co.uk/2012/06/public-data-sets-for-nialm.html. Accessed January 2020.

[13] G. Davis, S. Mallat, and M. Avellaneda. 1997. Adaptive Greedy Approximations. *Constructive Approximation* 13, 1 (March 1997), 57–98. https://doi.org/10.1007/BF02678430

[14] Egauge (2019). eGauge Energy Monitoring Solutions. http://egauge.net. Accessed January 2020.

[15] Frank Eichinger, Pavel Efros, Stamatis Karnouskos, and Klemens Böhm. 2015. A Time-Series Compression Technique and Its Application to the Smart Grid. *The VLDB Journal* 24, 2 (April 2015), 193–218.

[16] Google. 2019. Protocol Buffers Encoding: Signed Integers.

[17] Ramon Granell, Colin J. Axon, and David C. H. Wallom. 2015. Impacts of Raw Data Temporal Resolution Using Selected Clustering Methods on Residential Electricity Load Profiles. *IEEE Transactions on Power Systems* 30, 6 (Nov. 2015), 3217–3224. https://doi.org/10.1109/TPWRS.2014.2377213

[18] A. U. Haq, T. Kriechbaumer, M. Kahl, and H. Jacobsen. 2017. CLEAR – A circuit level electric appliance radar for the electric cabinet. In *Proceedings of the 2017 IEEE International Conference on Industrial Technology (ICIT '17).* 1130–1135.

[19] G.W. Hart. 1992. Nonintrusive Appliance Load Monitoring. *Proc. IEEE* 80, 12 (Dec. 1992), 1870–1891. https://doi.org/10.1109/5.192069

[20] D. A. Huffman. 1952. A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE* 40, 9 (Sep. 1952), 1098–1101.

[21] Ruoxi Jia, Fisayo Caleb Sangogboye, Tianzhen Hong, Costas Spanos, and Mikkel Baun Kjærgaard. 2017. PAD: Protecting Anonymity in Publishing Building Related Datasets. In *Proceedings of the 4th ACM International Conference on Systems for Energy-Efficient Built Environments* (Delft, Netherlands) *(BuildSys '17).* Association for Computing Machinery, New York, NY, USA, Article 4, 10 pages. https://doi.org/10.1145/3137133.3137140

[22] Richard Jumar, Heiko Maaß, and Veit Hagenmeyer. 2018. Comparison of Lossless Compression Schemes for High Rate Electrical Grid Time Series for Smart Grid Monitoring and Analysis. *Computers & Electrical Engineering* 71 (Oct. 2018), 465–476. https://doi.org/10.1016/j.compeleceng.2018.07.008

[23] Jack Kelly. 2015. UK Domestic Appliance Level Electricity (UK-DALE) - Disaggregated (6s) Appliance Power and Aggregated (1s) Whole House Power. https://doi.org/10.5286/UKERC.EDC.000001

[24] Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. 2001. Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases. *ACM SIGMOD Record* 30, 2 (June 2001), 151–162. https://doi.org/10.1145/376284.375680

[25] J. Zico Kolter and Matthew J. Johnson. 2011. REDD: A Public Data Set for Energy Disaggregation Research. In *SustKDD 2011.* San Diego, California.

[26] Thomas Kriechbaumer, Daniel Jorde, and Hans-Arno Jacobsen. 2019. Waveform Signal Entropy and Compression Study of Whole-Building Energy Datasets. In *Proceedings of the Tenth ACM International Conference on Future Energy Systems (e-Energy '19)* (Phoenix, AZ, USA) *(e-Energy '19).* Association for Computing Machinery, New York, NY, USA, 58–67.

[27] Thomas Kriechbaumer, Anwar Ul Haq, Matthias Kahl, and Hans-Arno Jacobsen. 2017. MEDAL: A Cost-Effective High-Frequency Energy Data Acquisition System for Electrical Appliances. In *Proceedings of the Eighth International Conference on Future Energy Systems (e-Energy '17)* (Shatin, Hong Kong). Association for Computing Machinery, New York, NY, USA, 216–221.

[28] D. Lemire and L. Boytsov. 2015. Decoding Billions of Integers per Second through Vectorization. *Software: Practice and Experience* 45, 1 (Jan. 2015), 1–29. https://doi.org/10.1002/spe.2203

[29] S.G. Mallat and Zhifeng Zhang. Dec./1993. Matching Pursuits with Time-Frequency Dictionaries. *IEEE Transactions on Signal Processing* 41, 12 (Dec./1993), 3397–3415. https://doi.org/10.1109/78.258082

[30] Antonio Notaristefano, Gianfranco Chicco, and Federico Piglione. 2013. Data Size Reduction with Symbolic Aggregate Approximation for Electrical Load Pattern Grouping. *IET Generation, Transmission & Distribution* 7, 2 (Feb. 2013), 108–117. https://doi.org/10.1049/iet-gtd.2012.0383

[31] Igor Pavlov. 1998. 7ZIP.

[32] S. R. Rajagopalan, L. Sankar, S. Mohajer, and H. V. Poor. 2011. Smart meter privacy: A utility-privacy framework. In *2011 IEEE International Conference on Smart Grid Communications (SmartGridComm).* 190–195.

[33] Ron Rubinstein, Alfred M Bruckstein, and Michael Elad. 2010. Dictionaries for Sparse Representation Modeling. *Proc. IEEE* 98, 6 (June 2010), 1045–1057. https://doi.org/10.1109/JPROC.2010.2040551

[34] Ron Rubinstein, Michael Zibulevsky, and Michael Elad. 2008. *Efficient Implementation of the K-SVD Algorithm Using Batch Orthogonal Matching Pursuit.* Technical Report CS-2008-08.revised. Technion Israel Institute of Technology, Haifa, Israel. 15 pages.

[35] Falk Scholer, Hugh E. Williams, John Yiannis, and Justin Zobel. 2002. Compression of Inverted Indexes For Fast Query Evaluation. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval - SIGIR '02.* ACM Press, Tampere, Finland, 222. https://doi.org/10.1145/564376.564416

[36] simdcomp (2019). SIMDComp: A simple C library for compressing lists of integers using binary packing. https://github.com/lemire/simdcomp. Accessed January 2020.

[37] Pecan Street. (2019). Dataport. https://dataport.pecanstreet.org/. Accessed January 2020.

[38] Michel P. Tcheou, Lisandro Lovisolo, Moises V. Ribeiro, Eduardo A. B. da Silva, Marco A. M. Rodrigues, Joao M. T. Romano, and Paulo S. R. Diniz. 2014. The Compression of Electric Signal Waveforms for Smart Grids: State of the Art and Future Trends. *IEEE Transactions on Smart Grid* 5, 1 (Jan. 2014), 291–302. https://doi.org/10.1109/TSG.2013.2293957

[39] J.A. Tropp. 2004. Greed Is Good: Algorithmic Results for Sparse Approximation. *IEEE Transactions on Information Theory* 50, 10 (Oct. 2004), 2231–2242. https://doi.org/10.1109/TIT.2004.834793

[40] Andreas Unterweger and Dominik Engel. 2016. Lossless Compression of High-Frequency Voltage and Current Data in Smart Grids. In *2016 IEEE International Conference on Big Data (Big Data).* IEEE, Washington DC,USA, 3131–3139. https://doi.org/10.1109/BigData.2016.7840968

[41] Andreas Unterweger, Dominik Engel, and Martin Ringwelski. 2015. The Effect of Data Granularity on Load Data Compression. In *Energy Informatics*, Sebastian Gottwalt, Lukas König, and Hartmut Schmeck (Eds.). Vol. 9424. Springer International Publishing, Cham, 69–80. https://doi.org/10.1007/978-3-319-25876-8_7

[42] Andreas Unterweger, Dominik Engel, and Martin Ringwelski. 2015. The Effect of Data Granularity on Load Data Compression. In *Proceedings of the 4th D-A-CH Conference on Energy Informatics - Volume 9424* (Karlsruhe, Germany) *(EI 2015).* Springer-Verlag, Berlin, Heidelberg, 69–80.

[43] U.S. Energy Information Administration. 2017. Nearly Half of All U.S. Electricity Customers Have Smart Meters.

[44] Yi Wang, Qixin Chen, Chongqing Kang, Qing Xia, and Min Luo. 2017. Sparse and Redundant Representation-Based Smart Meter Data Compression and Pattern Extraction. *IEEE Transactions on Power Systems* 32, 3 (May 2017), 2142–2151. https://doi.org/10.1109/TPWRS.2016.2604389

[45] Tri Kurniawan Wijaya, Julien Eberle, and Karl Aberer. 2013. Symbolic Representation of Smart Meter Data. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops on - EDBT '13.* ACM Press, Genoa, Italy, 242. https://doi.org/10.1145/2457317.2457357

[46] H. E. Williams. 1999. Compressing Integers for Fast File Access. *Comput. J.* 42, 3 (March 1999), 193–201. https://doi.org/10.1093/comjnl/42.3.193

[47] Shiyin Zhong, Robert Broadwater, and Steve Steffel. 2015. Wavelet Based Load Models from AMI Data. *arXiv:1512.02183 [cs]* (Dec. 2015). arXiv:1512.02183 [cs]

[48] M. Zukowski, S. Heman, N. Nes, and P. Boncz. 2006. Super-Scalar RAM-CPU Cache Compression. In *22nd International Conference on Data Engineering (ICDE'06).* IEEE, Atlanta, GA, USA, 59–59. https://doi.org/10.1109/ICDE.2006.150