Generalized Deduplication: Lossless Compression for Large Amounts of Small IoT Data

Rasmus Vestergaard, Daniel E. Lucani, and Qi Zhang
DIGIT and Department of Engineering
Aarhus University, Denmark
{rv, daniel.lucani, qz}@eng.au.dk

***Abstract*—We show that a generalization of deduplication can enable compressed storage of sensor data. The method uses error-correcting codes in a non-traditional manner to identify similar elements, and then leverages this similarity for compression. Using Reed Solomon codes, our method has a theoretical potential to reduce the cost of storing chunks of 16 bytes to as much as 5 times less, and up to 65 times less for chunks of 255 bytes. We define a simple model for sensor data, and show how our approach is able to compress data from the model, realizing its compression potential with much smaller data sets than classic deduplication requires. This demonstrates that generalized deduplication can be a viable solution for practical lossless compression of small IoT data in scenarios where classic deduplication is ineffective.**

***Index Terms*—Sensors, Compression, Storage, Deduplication**

I. Introduction

In the future the number of Internet of Things (IoT) devices is expected to increase massively. The ever-increasing amount of sensor data calls for novel solutions to store data efficiently. There are many strategies for compression of sensor data, and they can be classified as lossy [1] or lossless [2]. Although lossy compression can achieve higher compression ratios, it may cause undesirable loss of data precision, vital transient spikes, or other information. Therefore, before blindly applying lossy compression to achieve as much compression as possible, it should be evaluated whether losses are acceptable for the application [3][4]. For instance, an energy provider utilizing smart meters may require storage of the exact readings in order to be able to prove that consumers are billed correctly.

We propose a generalization of deduplication, and show how it can be used to facilitate lossless compression of sensor data. Deduplication is an established technique for compression in filesystems and other storage systems, and has been found to achieve significant compression in many practical scenarios [5]. The basic idea is to store each file chunk only once, and then simply replace subsequent copies with a pointer to the stored copy. However, if there is a minor difference between two chunks, then this similarity is not leveraged and both of the chunks will be stored in full. This might not be a good strategy if a data source has a tendency to generate chunks that are near-identical, which can be the case for IoT sensors. Even if they are sensing the same underlying physical process, their values are unlikely to be exactly equal due to factors such as time variance, spatial distribution, and noise. The advantage of our technique is that it allows chunks of near-identical data to be deduplicated, thus enabling compressed storage in the cloud while still providing independent retrieval of chunks and perfect reconstruction. Theoretical analysis of our scheme shows that it can converge faster than classic deduplication, requiring less data to reach a minimum cost per new chunk [6].

II. Generalized Deduplication

Classic deduplication is generalized to allow deduplication of near-identical file chunks, while still ensuring lossless reconstruction. Each chunk is decomposed into a *base* and a *deviation*, where the base contains the majority of information, and the deviation is "small". The intention is that two similar, but different, chunks can be represented by a single base and different deviations. Deduplication is applied to the bases, reducing the storage cost by storing each base only once. A many-to-one mapping is required to identify the base of a particular chunk. The image of the mapping is a set of bases, which is a subset of the chunks. The mapping takes a chunk as input, and output the base that is at the minimum distance from it. The difference between the base and the chunk can then be found, which we call the deviation. Clearly, it is possible to reconstruct the chunk using the base and the deviation. One option is to define this mapping through an error-correcting code. This will be detailed in the following section. We note that classic deduplication is obtained by considering each chunk as a base. The deviation is always the all-zero chunk and does not need to be stored, since the base fully specifies the original chunk.

A. Mapping from an error-correcting code

We now detail how an error-correcting code can be used to define the many-to-one mapping required for generalized deduplication. An error-correcting code is denoted $C_q(n,k)$, where k is the size of the messages, n is the size of the codewords, and $Q = 2^q$ specifies the Galois field that symbols are from, $\mathrm{GF}(Q)$. The messages are thus in the extension field $\mathrm{GF}\left(Q^k\right)$, where vector representation is used to consider each element a concatentation of k elements of the base field. The codewords are a subset of $\mathrm{GF}(Q^n)$. The codewords are the bases in generalized deduplication. An important metric for generalized deduplication is the *covering radius* of a code,

$$R(C) \triangleq \max_{\boldsymbol{v} \in \mathrm{GF}(Q^n)} \min_{\boldsymbol{c} \in C} d(\boldsymbol{v}, \boldsymbol{c}), \qquad (1)$$

TABLE I
THE BASE TABLE IN GENERALIZED DEDUPLICATION

base id	base
b_1	$[x_{1,1}x_{1,2}\dots x_{1,k}]$
b_2	$[x_{2,1}x_{2,2}\dots x_{2,k}]$
$\vdots$	$\vdots$
b_K	$[x_{K,1}x_{K,2}\dots x_{K,k}]$

TABLE II
THE CHUNK TABLE IN GENERALIZED DEDUPLICATION.

chunk id	base id	deviation
i_{c_1}	b_{c_1}	d_{c_1}
i_{c_2}	b_{c_2}	d_{c_2}
$\vdots$	$\vdots$	$\vdots$
i_{c_N}	b_{c_N}	d_{c_N}

where $d(\boldsymbol{v},\boldsymbol{c})$ denotes the Hamming distance between $\boldsymbol{v}$ and $\boldsymbol{c}$. Intuitively, the covering radius is the largest Hamming distance that any chunk $\boldsymbol{v}$ is from a valid codeword. When a chunk is ingested into the system, the closest base (codeword) is identified by decoding the chunk. The deviation is then found by comparing the base to the chunk. The set of bases and the deviations can be compressed independently.

Compression of the bases: Since the codewords of the code are used to form the set of bases, it is simple to represent them more compactly than the n symbols required for each chunk. Due to the nature of error-correcting codes, the set of codewords is defined by the message space of the code, so each base can be represented by k symbols by identifying the corresponding "message".

Compression of the deviations: Compressing the deviations requires slightly more thought. Since any chunk is at distance at most $R(C)$ from a base, this implies that the deviation has Hamming weight at most $R(C)$. This fact can be used to compress it. The deviation is represented as the location of the non-zero symbols, and the values of these symbols. The location can be represented with[1]

$$l_{\text{loc}}(C) = \left\lceil \log \sum_{r=0}^{R(C)} \binom{n}{r} \right\rceil \tag{2}$$

bits. The value must also be represented. For a Q-ary code, each symbol requires $\log Q = q$ bits. An exception to this is binary codes, where a non-zero value must be a 1, so it does not need to be stored. The cost of storing the symbols is thus

$$l_{\text{val}}(C) = \lceil \log (Q-1) \rceil R(C) \tag{3}$$

and the total cost of storing the deviation in compressed form comes from combining (2) and (3) as

$$l(C) = l_{\text{loc}}(C) + l_{\text{val}}(C) \quad \text{[bits]}. \tag{4}$$

B. Cost model

The total storage cost for generalized deduplication consists of two parts: The cost of representing the deduplicated bases, and the cost of storing each chunk. Assume that N chunks of n symbols, or qn bits, are stored using K distinct bases, and with the deviations requiring l bits each. The bases are organized in a table, as shown in Table I. The bases are all of the same size, k symbols or qk bits. Each of the K bases is assigned a $\lceil \log K \rceil$ bit identifier. Thus, each base requires

$$S_{\text{base}} = qk + \lceil \log K \rceil \tag{5}$$

[1]All logarithms in this paper are to base 2,
$\lceil x \rceil$ denotes rounding to the least integer greater than or equal to x.

bits to store. Another table contains the information about each chunk, as shown in Table II. Each of the N chunks are assigned a $\lceil \log N \rceil$ bit identifier, which they later can be retrieved by. For each chunk the base and deviation must also be indicated, causing each chunk to require

$$S_{\text{chunk}} = \lceil \log N \rceil + \lceil \log K \rceil + l \tag{6}$$

bits, where l is the number of bits used for the deviation. Finally, the total cost of storing the N chunks becomes

$$S_G = NS_{\text{chunk}} + KS_{\text{base}} \quad \text{[bits]}. \tag{7}$$

This should be compared to the cost of storing the same N chunks without applying the method. The files still require an id for retrieval, so the cost for each chunk becomes the qn bits of the chunk, and a $\lceil \log N \rceil$ bit identifier. The total becomes

$$S_B = N\left(qn + \lceil \log N \rceil\right) \quad \text{[bits]}. \tag{8}$$

This allows us to define the compression ratio,

$$G = \frac{S_B}{S_G}, \tag{9}$$

where $G > 1$ implies a compression. The goal of the method is to maximize G given a specific data configuration. An upper bound on the gain achievable for a particular code can be stated. Since a deviation must be stored for every chunk,

$$G < \frac{qn}{l(C)} \triangleq G_{\max}, \tag{10}$$

where qn is the number of bits required for storing the chunk's n symbols of q bits each. This bound is not achievable, since it only considers the cost of storing the deviation, and not the overhead incurred, but it can still be valuable to assess whether a particular code is worth pursuing.

III. Analyzing a code

This section analyzes the properties of Reed Solomon (RS) codes in the context of generalized deduplication. Reed Solomon codes are defined over a Galois field GF(Q). Let $n = 2^q - 1$. A code is then $C_q(n, n-2t)$ for some t, where t is the error-correction capability of the code. Since the codes are Q-ary, it is necessary to combine q bits to form each symbol. We use a symbol size of 8 bits, so each symbol corresponds to a byte. Equivalently, the RS code is defined over the Galois field GF(256). The RS code over this field is $C_8(255, 255-2t)$, which has a codeword length of $n = 255$ symbols. Since each symbol is a byte, this code will be able to handle chunks of 2040 bits when it is used for generalized deduplication. We can use a shortened version of this code [7] to have a more flexible

TABLE III
ANALYSIS OF A FEW REED SOLOMON CODES

Code	t	$R(C)$	$l(C)$	$G_{\max}$
$C_8(16,14)$	1	2	24	5.57
$C_8(16,12)$	2	5	53	2.42
$C_8(16,10)$	3	8	80	1.64
$C_8(64,62)$	1	2	28	18.96
$C_8(64,56)$	4	11	128	4.00
$C_8(64,48)$	8	16	178	2.89
$C_8(255,253)$	1	2	31	65.81
$C_8(255,247)$	4	8	113	18.05
$C_8(255,223)$	16	32	392	5.20

chunk size. This allows selection of a code fitting the chunk size for the data that must be compressed. In the following section we present simulations using a simple data model with 128 bit chunks, or 16 bytes. To fit this, we shorten the RS code to $C_8(16, 16-2t)$. It is also necessary to settle on what the error-correction capability of the code must be. This choice will have a large impact on the compression performance of our scheme. Larger correction capability means that each base has a shorter compressed representation, and thus requires less storage. Further, it will be more likely that chunks decode to the same base, which can increase the number of bases that are deduplicated. However, increasing the error-correction comes at the cost of a larger deviation representation, since the distance between the chunks and bases can be larger. This can easily become the dominating factor of the storage cost, so the error-correction capability should be carefully selected to optimize the storage cost.

Before turning to a specific data model, a brief analysis of a few specific codes is performed. To determine the number of bits required for representing the deviation, the covering radius of the code must be known. The covering radius is found by simulation, where MATLAB's implementation [8] of Reed Solomon codes is used. The simulation is simple: A chunk is generated by sampling and concatenating n i.i.d. uniformly distributed symbols from GF(256). This chunk is decoded to determine the corresponding "message". This message is then encoded, determining the base nearest the chunk. Finally, the Hamming distance between the chunk and the base is found. This process is repeated many times, and the largest distance observed is the covering radius. With enough iterations, it becomes unlikely that the true covering radius differs from the result of the simulation. It is possible to use (4) to calculate the storage cost of the deviation once the covering radius is known, which finally allows the upper bound to be assessed as in (10). Table III summarizes the results for a few different codes. In particular, it should be noted that codes for longer chunks can have a larger potential for gains in general.

IV. DATA MODEL AND REPRESENTATION

In order to study the performance of generalized deduplication, we define a simple data model. The assumption is that M nodes are distributed in some area, and are sensing some physical quantity, e.g., temperature. At times $i \in \{t_1, t_2, \ldots, t_N\}$ the nodes all perform the sampling. The nodes all send their

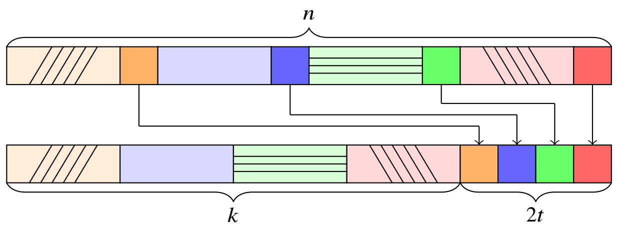

Fig. 1. Reordering to move high entropy bits to parity location

sensed value to an aggregator, where the values are combined to form a single chunk. The sensed value of each node is modeled as a Gaussian distributed random variable,

$$C_{i,j} \sim N(\mu_j, \sigma_j^2), \quad j \in \{1, 2, \ldots, M\}, \tag{11}$$

sampled independently from the other nodes. Each value is represented in floating point format with 32 bits. The aggregation is then modeled by concatenating the value of each of the M nodes as

$$C_i = \left[C_{i,1}, \ldots, C_{i,M}\right], \tag{12}$$

obtaining a chunk of $32M$ bits. Identical distributions will be assumed for the nodes, although this is not a requirement. With $M = 4$, the data model is

$$C_i \sim \left[N(\mu, \sigma^2),\ N(\mu, \sigma^2),\ N(\mu, \sigma^2),\ N(\mu, \sigma^2)\right], \tag{13}$$

where μ is the common mean and σ the common standard deviation. This model has chunks of 16 bytes, and will be used for subsequent simulations.

Our preliminary simulations with the shortened RS code indicated that matching bases are likely to occur when chunks only differ in the symbols that are traditionally considered the parity. This fact is exploited in order to maximize the number of matching bases. Before applying generalized deduplication, a reordering of the bit order in all of the chunks were used. By moving the bits with highest entropy to the parity location, the likelihood of matches is maximized. In the aggregator model presented above, the bits with the highest entropy are the least significant bits of each floating point value. These bits will essentially be equally likely to be a 0 or 1, independent of the other bits in the value. As an example, if the systematic $C_8(16,14)$ code is used, there are two bytes of parity at the end. This means that the four least significant bits of each measurement can be moved to the end of the chunk. The reordering is illustrated in Figure 1. As the reordering process is always the same for a particular code, it can be reverted during data retrieval.

V. NUMERICAL RESULTS

Simulation results for the data model in (13) can now be presented. We start by comparing classic deduplication and the generalization with $C_8(16,14)$. Initially the standard deviation is kept small to allow deduplication to happen with relatively small amounts of data. In particular, values are drawn i.i.d. from a Gaussian distribution with $\mu = 20$ and $\sigma = 5 \cdot 10^{-6}$.

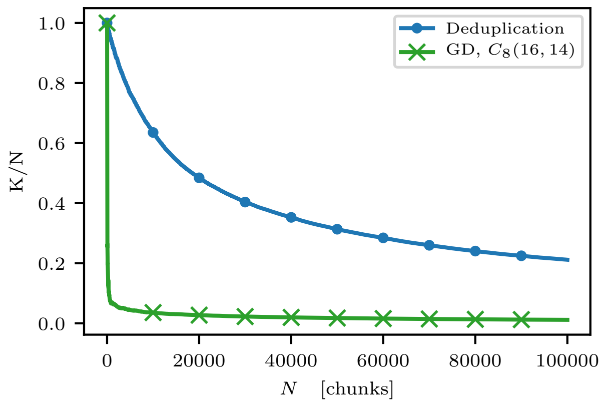

Fig. 2. The number of bases in deduplication and the generalization.

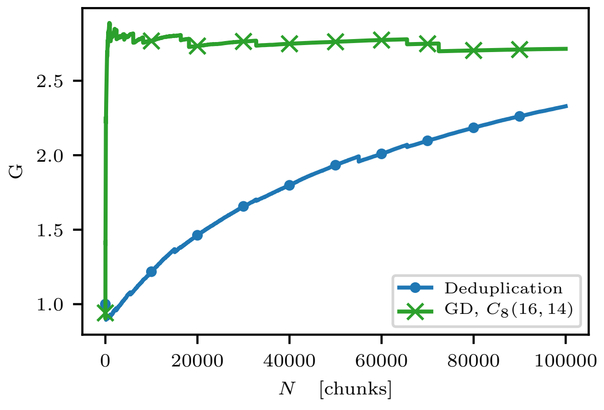

Fig. 4. The compression ratio for deduplication and the generalization.

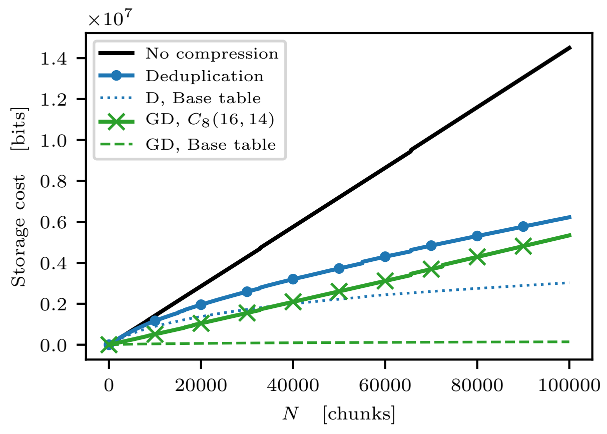

Fig. 3. Total storage costs and base table storage cost for deduplication and the generalization. The total storage cost contains both chunk- and base table.

The entropy of one element from this distribution as a 32 bit floating point number is estimated to be 6.76 bits. The entropy of a chunk with four concatenated values is thus approximately 27 bits. According to Shannon's source coding theorem [9], an expected compression ratio of 4.74 is the highest that can be achieved. Figure 2 shows the fraction of bases to chunks. A smaller value is desirable, since it indicates that common bases for chunks are found. When the entire set of bases is identified, all subsequent chunks can be deduplicated, so the expected cost of each chunk will be at the minimum. It is clear that the generalization is able to quickly find the set of common bases that is used to describe the chunks. Since common bases only are found in classic deduplication when two chunks are identical, it takes much longer time to discover the set of bases. As more data is processed it will eventually happen, although the amount of chunks must be much greater. This is in line with the theoretical analysis in [6], where it was shown that the generalization can converge to a minimum cost for subsequent chunks much faster than classic deduplication. The main question is whether the smaller number of bases does indeed outweigh the additional overhead and translate to a better compression. The total storage cost from (7) is therefore computed for the simulation. The storage cost consists of two parts: The bases, and the chunks' metadata including the deviation. Figure 3 shows the overall cost for the two methods, as well as the cost of storing the data with no compression. Generalized deduplication is able to achieve a lower storage cost than classic deduplication. This is largely caused by the much smaller cost of storing the appropriate bases, as the cost per chunk is larger. This is reflected in the storage cost of the base tables. The gap between the overall cost and the base table is the cost of storing the chunk table. Classic deduplication needs more bases, and the cost of storing them is dominant through most of the figure. As more data is ingested, near the end of the figure the cost of storing the chunk data also becomes dominant for the classic method. Most of the bases are already identified, so subsequent chunks tend to add only the cost of its entry in the chunk table. Figure 4 shows the compression ratio. Applying either of the two deduplication methods does achieve compression gains for this data configuration. Neither of the methods approaches the theoretical bound, but this was not expected to be the case for our setup, since additional overhead is introduced for storage. The potential gains of the generalization are realized earlier, which can be important in practice. When the amount of data available is not large enough to achieve significant gains with classic deduplication, it may already be possible to compress the data with the generalization.

The simulation is repeated with a large number of configurations. In particular, a range of standard deviations and number of chunks is studied. The results are summarized in Figure 5, where it is seen how classic deduplication compares to three variants of the generalization, using the three shortened RS codes with $n = 16$ from Table III. The gain and range of applicability varies between the variants. For instance, $C_8(16,14)$ achieves a gain when the standard deviation is below 10^{-4}, and it achieves its best compression ratio of 3 when the standard deviation is close to 10^{-6}. $C_8(16,10)$ can achieve a gain as long as the standard deviation is below $2 \cdot 10^{-2}$, but the gain is smaller with its maximum compression ratio only slightly above 1.32. Although classic deduplication can achieve the best overall compression ratio of more than 5, this occurs only at extremely small standard deviations as low as 10^{-6}. This is where the standard deviation is small enough that

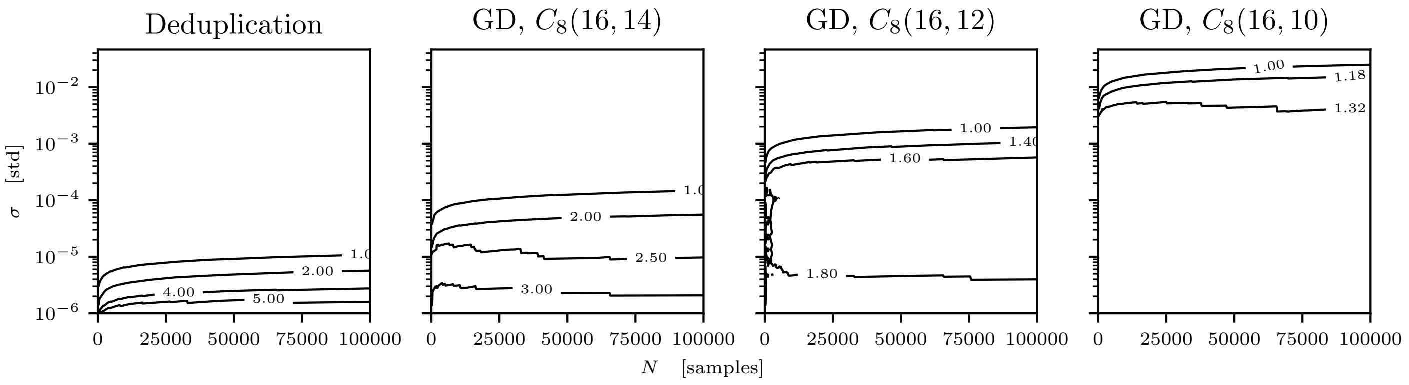

Fig. 5. The compression ratios achieved by generalized deduplication. Some lines of equal gain are shown.

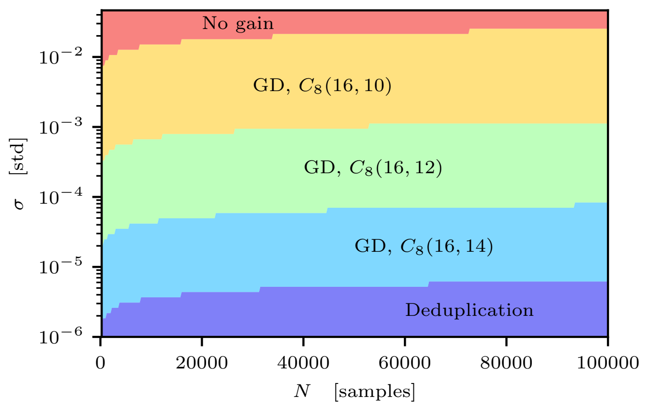

Fig. 6. Indication of which strategy that achieves the best compression for each data configuration

only few distinct values are output by the quantized normal distribution. With a large enough data set it will be impossible for the generalization to achieve a better compression than the classic approach, where the overhead is smaller. However, if the chunks have reasonably high entropy, it is unlikely that many exact matches happen, since the number of potential chunks is so large. The advantage of generalized deduplication is to be able to achieve a compression gain over a wider range of standard deviations and data set sizes. An alternative view on this is seen in Figure 6, where the configuration with the best compression ratio is indicated. Clearly, there are many scenarios in which the generalization can be useful, so it poses an opportunity for reducing the storage cost of IoT data.

VI. Conclusion

The ever-increasing amounts of sensor data generated by the IoT calls for novel solutions to store the data efficiently. We have shown that generalized deduplication can be used to deduplicate data sets with chunks that are similar but not identical, which has the potential to achieve compression gains for a range of data sets where classic deduplication is ineffective. Our studies use a model for IoT data with small data chunks of only 128 bits. Larger chunks should be used in practice, since this can increase the potential of the method as the overhead will be less significant. For the model in this paper, we have observed compression ratios of 1.3-2.5 for configurations where classic deduplication would have expanded the data rather than compressing it. The generalization is able to identify matches and deduplicate them with much fewer data chunks than the classic approach. This offers an opportunity for lossless compression and compressed storage when the amount of data is limited and smaller than what would be required to achieve gains with classic deduplication.

Our future work will investigate how the method scales with increasing chunk size and number of chunks. The method will also be applied to real IoT data sets, in order to evaluate the true compression benefits. Further investigation is required to explore other options in terms of mappings, e.g., other codes may prove more suitable in practical scenarios. It is also worth studying how to optimize the compression gain by intelligently ordering data based on its characteristics.

Acknowledgment

This work was partially financed by the SCALE-IoT project (Grant No. DFF - 7026-00042B) granted by the Danish Council for Independent Research, the Aarhus Universitets Forskningsfond Starting Grant Project AUFF-2017-FLS-7-1, and Aarhus University's DIGIT Centre.

References

[1] T. Bose, S. Bandyopadhyay, S. Kumar, A. Bhattacharyya, and A. Pal, "Signal Characteristics on Sensor Data Compression in IoT -An Investigation," in *IEEE SECON 2016*, jun 2016.

[2] J. Spiegel, P. Wira, and G. Hermann, "A Comparative Experimental Study of Lossless Compression Algorithms for Enhancing Energy Efficiency in Smart Meters," in *IEEE INDIN 2018*, jul 2018.

[3] N. Hübbe, A. Wegener, J. M. Kunkel, Y. Ling, and T. Ludwig, "Evaluating Lossy Compression on Climate Data," in *Supercomputing*. Springer, 2013, pp. 343–356.

[4] A. Moon, J. Kim, J. Zhang, H. Liu, and S. Woo Son, "Understanding the impact of lossy compressions on IoT smart farm analytics," in *IEEE Big Data 2017*, dec 2017, pp. 4602–4611.

[5] W. Xia, H. Jiang, D. Feng, F. Douglis, P. Shilane, Y. Hua, M. Fu, Y. Zhang, and Y. Zhou, "A Comprehensive Study of the Past, Present, and Future of Data Deduplication," *Proc. IEEE*, vol. 104, no. 9, pp. 1681–1710, 2016.

[6] R. Vestergaard, Q. Zhang, and D. Lucani, "Generalized Deduplication: Bounds, Convergence, and Asymptotic Properties," *arXiv:1901.02720 [cs.IT]*, Jan 2019.

[7] J. Castiñeira Moreira and P. G. Farrell, *Essentials of Error-Control Coding*. Wiley, 2006.

[8] "MATLAB and Communications Toolbox," The MathWorks, Inc., Natick, MA, USA, 2018.

[9] T. M. Cover and J. A. Thomas, *Elements of information theory*. Wiley-Interscience, 2006.