Combined compression of multiple correlated data streams for online-diagnosis systems

Simon Meckel[a], Markus Lohrey[a], Seungbum Jo[b], Roman Obermaisser[a], Simon Plasger[a]

[a] *Department of Electrical Engineering and Computer Science, University of Siegen, Germany*
[b] *Department of Computer Science, Chungbuk National University, Korea*

Abstract

Online fault-diagnosis is applied to various systems to enable an automatic monitoring and, if applicable, the recovery from faults to prevent the system from failing. For a sound decision on occurred faults, typically a large amount of sensor measurements and state variables has to be gathered, analyzed and evaluated in real-time. Due to the complexity and the nature of distributed systems all this data needs to be communicated among the network, which is an expensive affair in terms of communication resources and time. In this paper we present compression strategies that utilize the fact that many of these data streams are highly correlated and can be compressed simultaneously. Experimental results show that this can lead to better compression ratios compared to an individual compression of the data streams. Moreover, the algorithms support real-time constraints for time-triggered architectures and enable the data to be transmitted by means of shorter messages, leading to a reduced communication time and improved scheduling results. With an example data set we show that, depending on the parameters of the compression algorithm, more than one third of the bits (34.3 %) in the data communication can be saved while only on about 0.2 % of all data values a slight loss of accuracy occurs. This means 99.8 % of the data values can be correctly delivered without any loss but with a significant reduction of bandwidth demands.

Keywords: Online-diagnosis, Real-time, Data compression, Scheduling

1. Introduction

It is the goal of online fault-diagnosis to detect, diagnose and, especially in safety-critical application domains, overcome system faults at run time, e.g., through reconfigurations, in order to provide the required services with a very high reliability. Typical faults are faults in the design of a component or a system, transient or permanent hardware faults or erroneous user operations amongst others. As faults cannot be prevented at all times it is of uttermost importance to equip the system with powerful diagnostic algorithms able to process, analyze and store the often huge amount of raw data of a variety of sensors as well as system input and output parameters necessary for correct and fast decisions on fault detection and identification [1, 2].

The quality of diagnostic decisions highly depends on the available amount of data. Especially in distributed systems where the diagnostic inference process is temporally and spatially decomposed to multiple processing units, a limited storage capacity of local real-time databases may narrow the performance of diagnostic decisions, e.g., of long term trend analyses. Consider for example distributed sensor networks as introduced in [3]. Likewise, limited communication resources or bottlenecks may require data to be discarded or may lead to communication delays, which are also disadvantageous with respect to the time needed to conclude a specific fault from a first symptom. This, however, is a quality characteristic of a diagnostic architecture. There is also a trend towards the application of neural networks and deep learning for fault-diagnosis, where especially architectures with multiple neural networks might be optimally executed on distributed systems, see e.g., [4, 5].

Since a distributed diagnostic process requires both, the storage and the fast exchange of a huge amount of data within a network, the DAKODIS[1] project deploys data compression for online fault-diagnosis in a time-triggered architecture. In [6] it is shown that data compression is a feasible instrument to save bandwidth and consequently provide stronger real-time guarantees or save communication resources (see [7] for an introduction into the wide area of data compression). Especially systems with limited resources may require data compression to actually make fault-diagnosis possible or keep a diagnosis system alive if more and more traffic is scheduled to the existing communication channels.

The majority of compression algorithms such as entropy-based compressors or dictionary-based compressors (see e.g., [7]) are not suitable for time-triggered systems, as they do not guarantee a fixed compression ratio; see also Section 2. This was overcome in [6], where a simple cache-based compression algorithm was presented that allows the loss of a small ratio of the input data values. It works especially well if the input data stream shows a temporal locality in the sense that consecutive data values are close with high probability. Physical sensor signals typically show this behavior.

In [6], only the compression of a single data stream is addressed. On the other hand, diagnostic data streams within distributed networks are often highly correlated. For instance,

[1] DAKODIS – Data compression for online-diagnosis systems. The primary objective of this research project is an increased efficiency and the reduction of overhead for online-diagnosis in open embedded systems using data compression. (see `https://networked-embedded.de/es/index.php/dakodis.html`)

multiple redundant sensors measure the same physical quantities, or measurements of interdependent components or subsystems show other kinds of characteristic dependencies. These correlations yield the potential for further compression. In time-triggered architectures all routes for the messages are predetermined. In [8] it is demonstrated that if messages of multiple data streams that take the same routes can be compressed to one combined sample, scheduling results can be improved. However, no real-time compression algorithms for such a scenario have been introduced so far. In this paper we close this gap and extend the work from [6] to the simultaneous compression of multiple data streams. We evaluate the resulting compressor with voltage and current measurement signals from a DCDC converter and demonstrate that the simultaneous compression of multiple correlated data streams leads to a better compression ratio without increasing the overall number of lost data values. The proposed compressors support real-time constraints for time-triggered architectures by maintaining a constant compression ratio. For the identification of potential candidates for a combined compression, a pre-analysis of the available data streams (and their source and destination) in the system is conducted. However, the compression algorithm itself has no a-priori knowledge about the signal behaviors. The results of the compression algorithm provide a scheduler with meaningful information to plan the most beneficial task allocations and message combinations.

This paper is an extended version of our DSD 2019 conference paper [9]. The new contributions include improvements to the introduced online compression algorithm, which now supports an automatic detection of changes of signal characteristics, and thus, offers a significantly improved ability to exploit multiple correlated signals for a combined compression and transmission. Moreover, we show that the selection of suitable signals for the combined compression can now be automatically handled by the algorithm, which allows to optimally set up the compression parameters without manual intervention of system operators.

The paper is organized as follows: In Section 2 we refer to the overall system architecture and the specific requirements for the compression algorithm. Section 3 then focuses on the principles and implementations of the proposed compression algorithm. The evaluation is conducted in Section 4 by means of an example model and signals. The paper closes with a future work section and a conclusion in Sections 5 and 6, respectively.

2. Model

An architecture for time-triggered real-time systems is defined in [6]. The three core aspects of the model are (1) logical modeling of diagnostic processing tasks, (2) intertask communication procedure modeling supporting data compression and (3) task scheduling to map the processing tasks to computation nodes of a physical network infrastructure, and thus, establishing a time-triggered architecture (see [10] for an introduction to timing-predictable embedded systems). Within this environment, specific requirements for the compression algorithm arise:

- Compression and decompression underlie hard real time constraints.
- Only online (one-pass) compression algorithms can be used that compress every arriving data value before the next data value arrives.
- No statistical information concerning the probability distribution of the data values is available.
- In order to utilize compression for the scheduling process, the compressor should guarantee a certain worst-case compression ratio below one.
- Compression should be lossless (after an initial quantization phase for sensor data), with the exception that occasionally data values can be completely lost. This means that every data value is either transformed by the sender into a compressed representation that can be exactly recovered by the receiver, or it is transformed into an indication value that tells the receiver that the original data value is lost and can only be reconstructed with a lower accuracy. Losing some data values is unavoidable if we want to guarantee a worst-case compression ratio below one. A small probability for losing data values is tolerable in our context, since diagnosis is typically not dependent on single data values.

These requirement rule out most of the classical compressors, e.g., lossy compressors based on transform coding, as they are inherently lossy and do not allow to recover data values without error and also lossless compressors (e.g., entropy-based coders, or arithmetic coding) as they cannot guarantee a fixed compression ratio. For a detailed overview on classical compression techniques see [7].

For our scenario we assume the data communication to be error free, i.e., a data value is received exactly as it was sent. Under challenging conditions, forward error correction is a feasible instrument to ensure this.

3. Compression

The compression schemes proposed in this paper are based on the work of [6], where the overall system architecture as well as the compression model is addressed in detail. Therefore, Section 3.1 briefly summarizes the working principles of the cache-based online data compression algorithm before the subsequent sections deal with the extension to the compression of multiple correlated data streams.

3.1. Compression of individual data streams

The goal is to compress a sequence of n-bit data values (for a fixed n), which are produced by a sensor measuring a physical quantity. It is assumed that the data values exhibit some locality. For the compression every n-bit data value is split into a block of s high-order bits (called the *head*) and the remaining $t = n - s$ low-order bits (called the *tail*). Due to the locality of consecutive data values, the heads of consecutive data values

are expected to only show a small variation over time. The $n = s + t$ bits in a data value are compressed to $r + t$ bits (for some $r < s$). For this, the algorithm transmits the t bits from the tail uncompressed and compresses the s bits from the head to r bits. To achieve the latter, a dictionary D stores the $2^r - 1$ most recently seen heads at dictionary entries $D[p]$, where the dictionary index p is an r-bit code different from the reserved sequence 0^r (r 0-bits). Let the next n-bit data value be $x = uv$, where $u \in \{0,1\}^s$ is the head and $v \in \{0,1\}^t$ is the tail. If the head u is in the dictionary and stored at entry $D[p]$ then the $(r + t)$-bit code pv is transmitted. Otherwise, a so called *miss* occurs which is indicated to the receiver by the bit sequence 0^r. The sender then transmits the bit sequence $0^r u$ (recall that $s \leq t$ so $0^r u$ fits into $r + t$ bits). Moreover, sender and receiver update their dictionaries by computing a dictionary index $p = \text{fresh}(D)$ and setting $D[p] := u$. Here $\text{fresh}(D)$ is the index of a free dictionary entry, or, if the dictionary is completely filled, an index that is computed by some replacement strategy (we use the least-recently-used strategy, LRU for short). The remaining $t - s$ bits following $0^r u$ can be used to transmit the $t-s$ most significant bits (MSBs) of the tail v to achieve a higher accuracy. Due to locality in the data values we expect a small number of misses over time. Note that sender and receiver can keep their dictionaries synchronized. The main features of the compression algorithm are:

- A fixed compression ratio of $(r + t)/(s + t) < 1$ is achieved for every data value. This is contrary to classical lossless compression, where only statements about the average compression ratio are possible. However, a fixed compression ratio is important for time-triggered architectures.
- To make a fixed compression ratio < 1 possible, we have to accept occasional losses of data values. A small number of lost data values is acceptable for many online-diagnosis applications. Moreover, even in the case of a miss, an approximation in form of the t MSBs of the data value is transmitted.
- Those data values that are not lost are transmitted without any loss in accuracy, which is in contrast to classical lossy compression.

Related work from data compression. The idea of using locality in the data values for compression can be found in many works. One of the simplest ways of exploiting locality is delta-coding, where differences between consecutive data values are transmitted. These differences are typically small and can be further compressed with an entropy encoder. In the context of wireless sensor networks this idea is implemented in the LEC-compressor from [11, 12]. In contrast to our method, the LEC-compressor is a lossless entropy-based encoder, which does not show a fixed compression rate. In lossy compression, differential encoding [7, Chapter 11] exploits correlation between successive data values by transmitting the differences between a prediction of the next data value and the actual data value. In the area of information theory the problem of compressing correlated data streams is known as distributed source coding, see [13].

3.2. Simultaneous compression of multiple data streams

The cache-based compression algorithm introduced in Section 3.1 handles each data stream individually. For the compression it utilizes the fact that measurements of physical quantities can be often covered by just a part of the overall code word space for certain time intervals. With a view to the overall system architecture, many data streams for monitoring and diagnostic purposes (e.g., voltage, current or vibration measurements) of complex mechatronic systems are gathered and processed at different locations and need to be exchanged via a distributed network. Often, many of these data streams are highly correlated due to redundant measurements or physical relations of the measured signals. The enhanced compression scheme presented in the following takes advantage of both facts, the neighborhood assumption and the signal correlations. Since the overall data traffic needs to be scheduled in time-triggered architectures, the transmission routes are predetermined. Especially when the source and the destination nodes of two or more data streams are located close to each other or are the same, the overall message size for transmitting the data can be reduced, leading to advantageous scheduling results (e.g., a shorter makespan).

To exploit correlations of data streams we adapt the cache-based encoding scheme to encode multiple data streams at once. Assume that we have d data streams. Let x_i ($1 \leq i \leq d$) be the current data value of the i-th stream. We further assume that x_i is an n_i-bit data value. For every $i \in [1, d]$ we fix a partition $n_i = s_i + t_i$ and split x_i into $x_i = u_i v_i$ with $|u_i| = s_i$ and $|v_i| = t_i$. The bit sequence u_i (resp., v_i) is the current head (resp., tail) of the i-th stream. We do not assume $s_i = s_j$ or $t_i = t_j$ for $i \neq j$. Let $s = \sum_{i=1}^{d} s_i$ and $t = \sum_{i=1}^{d} t_i$ for the rest of the section. One could apply the compression scheme from Section 3.1 to each of the d data streams separately by choosing numbers $r_i < s_i$ and maintaining a dictionary of size $2^{r_i} - 1$ for every $1 \leq i \leq d$. This leads to an overall compression ratio of $(\sum_{i=1}^{d} r_i + t)/(s + t)$. On the other hand, due to correlations between the data streams, the tuples of data values $(x_1, \ldots, x_d)$ will be scattered around a low dimensional subspace of the d-dimensional product space. If, for instance, $d = 2$ and $x_2 = f(x_1)$ for a function f then all tuples belong to a one-dimensional curve in the two-dimensional plane. In such a case we can obtain a better compression ratio of $(r + t)/(s + t)$ by using a single dictionary of size $2^r - 1$ for some $r < \sum_{i=1}^{d} r_i$ that stores tuples of heads $u = (u_1, \ldots, u_d)$, which need s bits. The compressed data value then consists of the dictionary index $p \in \{0,1\}^r$, where u is stored and the concatenation $v_1 v_2 \cdots v_d$ of the current tails. In case of a miss we transmit 0^r followed by u. This leaves $t - s$ unused bits. Likewise to the original approach we use them to transmit parts of the tails. In the multidimensional case there are several possibilities:

- Transmit as many tails as possible completely and fill the remaining bits with the MSBs of the next tail,
- transmit equally many MSBs for each tail,
- transmit MSBs for each tail; the number of MSBs is weighted by the complete tail length.

Note that in this approach all d data values from $(x_1, \ldots, x_d)$ are lost in case of a miss. In consequence, this method is a trade-off between a better (i.e., lower) compression ratio and a higher number of lost data values. Its implementation with a variable number d of streams can be found on our project website.

Algorithm 1 Cache-based algorithm for d streams

input : data values $x_i \in \{0,1\}^{n_i}$ $(1 \le i \le d)$
output : bit string of length at most $r + t$
initialize dictionary D as empty hash table of size $2^r - 1$
let $x_i = u_i v_i$ with $|u_i| = s_i$ and $|v_i| = t_i$ for $1 \le i \le d$
if there is p with $D[p] = (u_1, \ldots, u_d)$ **then**
    send $pv_1v_2 \cdots v_d$ to the receiver
else
    send $0^r u_1 u_2 \cdots u_d$ to the receiver
    $p :=$ fresh(D)
    $D[p] := (u_1, \ldots, u_d)$
end if

3.3. Dynamic simultaneous compression of multiple data streams

It turns out that one can reduce the number of misses by adding some flexibility using offsets in the dictionary entries. Consider a sequence of values $S = a_1, a_2, \ldots, a_k$ on a single stream such that for all $1 \le i < k$, $|a_i - a_{i+1}| < 2^{t-1}$ but $\lfloor a_i/2^t \rfloor \ne \lfloor a_{i+1}/2^t \rfloor$ (s and t are the size of the head and tail, respectively). If one compresses S using the cache-based compression algorithm from Section 3.1 with a dictionary of size 1 (i.e., $r = 2$), then this results in k misses, since there are no consecutive values in S with the same head. To handle this case, Jo et al. proposed a *dynamic cache-based algorithm* [6]. In this algorithm, every dictionary entry $D[p]$ is a pair (u, δ) of a head $u \in \{0,1\}^s$ and an *offset* $\delta \in [-2^{t-1}, 2^{t-1} - 1]$. We define a corresponding interval $I[p] = [u2^t + \delta, u2^t + \delta + 2^t - 1]$. Here and in the following we identify heads (resp., tails) with numbers from the interval $[0, 2^s - 1]$ (resp., $[0, 2^t - 1]$) using their binary representation. We say that p *covers* the data values in the interval $I[p]$. The cache-based compression algorithm from Section 3.1 can be considered as the special case where δ is always 0. Let us write $(u[p], \delta[p])$ for $D[p]$. When the sender transmits a code word $x = uv$ where u (resp. v) is the head (resp., tail) of x, the sender first checks whether there is a dictionary index p that covers x. Since $\delta[p] \in [-2^{t-1}, 2^{t-1} - 1]$, we must have $u[p] \in \{u-1, u, u+1\}$. Now suppose that the dictionary contains an index p which covers x (otherwise, the sender transmits $0^r u$ and adds the pair $(u, 0)$ to the dictionary). The sender takes the smallest such p and transmits pv to the receiver. Then it updates the dictionary in such a way that x becomes the center of an interval $I[q]$ for some dictionary index q. For this, it first ensures that $u[q] = u$ holds for a unique index q. Then, it sets the offset $\delta[q]$ to $v - 2^{t-1}$. In this way, x becomes the center of the interval $I[q]$. The receiver reconstructs x from p and v and updates its dictionary analogously. One easily observes that in the above example only a_1 is lost with this dynamic cache-based compression algorithm while maintaining a dictionary of size 1.

To compress highly-correlated multiple streams efficiently when each stream has consecutive data values with small gaps, we make Algorithm 1 dynamic: Each dictionary entry $D[p]$ stores a d-tuple $(u_1, \ldots, u_d)$ of heads $(u_i \in \{0,1\}^{s_i})$ and an offset vector $(\delta_1, \ldots, \delta_d)$ with $\delta_i \in [-2^{t_i-1}, 2^{t_i-1} - 1]$. Let us define $u[p, i] = u_i$, $\delta[p, i] = \delta_i$ and the interval

$$I[p, i] = [u_i 2^{t_i} + \delta_i, u_i 2^{t_i} + \delta_i + 2^{t_i} - 1].$$

We say that p covers the data values in the d-dimensional hypercube

$$H[p] := \prod_{i=1}^{d} I[p, i]. \tag{1}$$

We call this hypercube an *active hypercube* and the union of all active hypercubes is called the set of *active data tuples*. Now consider the case that the sender transmits a tuple $x = (x_1, \ldots, x_d)$ of data values from d data streams to the receiver. Let $x_i = u_i v_i$ as in Section 3.2. Then the sender first checks whether there is a dictionary index p that covers x (otherwise, the sender transmits $0^r u_1 \cdots u_d$, and adds $(u_1, \ldots, u_d)$ with the offset vector $(0, \ldots, 0)$ to the dictionary). For every $1 \le i \le d$ we must have $u[p, i] \in \{u_i - 1, u_i, u_i + 1\}$. The sender again takes the first such p and transmits $pv_1v_2 \cdots v_d$ to the receiver. Then it makes the same updates that were described above for the case $d = 1$ in every dimension, see Algorithm 2.

Algorithm 2 Dynamic cache-based algorithm

input : data values $x_i \in \{0,1\}^{n_i}$ $(1 \le i \le d)$
output : bit string of length at most $r + t$
miss : variable for indicating the event of a miss
initialize dictionary D as empty hash table of size $2^r - 1$
let $x_i = u_i v_i$ with $|u_i| = s_i$ and $|v_i| = t_i$ for $1 \le i \le d$.
if there is $p \in \{0,1\}^r \setminus \{0^r\}$ covering $(x_1, \ldots, x_d)$ **then**
    let p be the smallest index covering $(x_1, \ldots, x_d)$
    send $pv_1 \cdots v_d$ to receiver
    miss := 0
else
    send $0^r u_1 u_2 \cdots u_d$ to receiver
    miss := 1
end if
if there is no q with $u[q] = (u_1, \ldots, u_d)$ **then**
    $q :=$ fresh(D)
    $u[q] := (u_1, \ldots, u_d)$
    $\delta[q] := (0, \ldots, 0)$
end if
if miss = 0 **then**
    let q be the unique index with $u[q] = (u_1, \ldots, u_d)$
    $\delta[q] := (v_1 - 2^{t_1-1}, \ldots, v_d - 2^{t_d-1})$
end if

In the following two subsections we explain the improvements of Algorithm 2 that further reduces the number of lost data values considerably.

3.4. Partial misses

When compressing d data streams simultaneously, a miss occurs if there is no p covering x. Consider an example with $d = 2$ streams where $s_1 = s_2 = 4$, $t_1 = t_2 = 8$, and $r = 2$, i.e., the dictionary D is of size $2^r - 1 = 3$. Table 1 shows an example for a dictionary; all offsets are assumed to be zero. The dictionary

Table 1: Codebook for two-dimensional stream compression

dictionary index p	$u[p,1]$	$u[p,2]$
01	0110	1010
10	0111	1011
11	1001	1100

index 00 is reserved for the indication of the miss case. If the heads of the current data values x_1 and x_2 are $u_1 = 1001$ and $u_2 = 1101$, respectively, there is a miss as the required concatenation of heads is not in the dictionary. Hence, the tail bits are used to send the new head combination u_1u_2 to the receiver. Figure 1 illustrates the active hypercubes (here squares) as blue squares with the missed data value marked as a red cross.

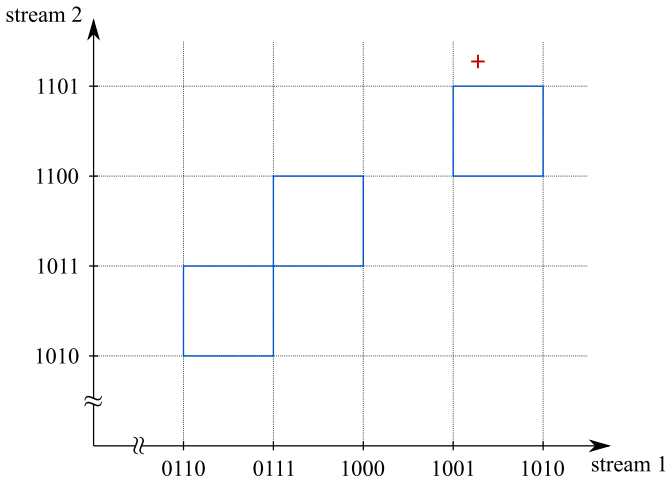

Figure 1: Simultaneous compression of two data streams

In the above situation, both data values x_1 and x_2 are lost. The strategy presented in the following overcomes this disadvantage of the algorithm in many cases and allows to communicate the current data values corresponding to some of the streams successfully. This is realized by reserving an additional dictionary index (the bit sequence $0^{r-1}1$) to indicate a so called *partial miss* (in contrast to a *full miss* which is indicated by 0^r). Hence, the dictionary size is reduced to $2^r - 2$. The number of bits transmitted per time step is $r + t$ (the length of the dictionary index plus the length of all tails) as before. In the case of a partial miss, the bit sequence following $0^{r-1}1$ is interpreted in a way different to our previous algorithm. Given the example of Table 1, $D[01]$ is removed and $p = 01$ is used to indicate a partial miss.

If there is no dictionary index p that covers the whole tuple of data values $(x_1, \ldots, x_d)$, the algorithm seeks for a p that covers at least some of the data values. For this, we define the set $M[p] = \{i \in [1,d] \mid x_i \notin I[p,i]\}$ of those dimensions where p does not cover the corresponding data value. In our example ($u_1 = 1001$ and $u_2 = 1101$) we have $M[10] = \{1,2\}$ and $M[11] = \{2\}$.

Our algorithm tries to encode as many entries from the input tuple $(x_1, \ldots, x_d)$ as possible by looping over all dictionary indices $p \in \{0,1\}^r \setminus \{0^r, 0^{r-1}1\}$. Consider a specific index p. Assume that we tell the receiver p and the set $M[p]$. This leaves $t - r - d$ bits from the initial $r + t$ bits (we need r bits for the partial miss indicator $0^{r-1}1$, r bits for p and d bits for $M[p]$). We can use these $t - r - d$ bits in order to send the following data to the receiver:

1. All tails v_i for $i \in [1,d] \setminus M[p]$. Note that for $i \in [1,d] \setminus M[p]$, the receiver can obtain the head u_i of x_i (and consequently the data value x_i) from the dictionary entry $D[p]$ as in Section 3.3. Note that $\sum_{i \in [1,d] \setminus M[p]} t_i$ bits are needed for these tails.
2. All differences $u_i - u[p,i]$ for every dimension $i \in M[p]$. Note that for each difference we need one bit to encode the sign of the difference. Also note that the differences $u_i - u[p,i]$ ($i \in M[p]$) together with the index p allow to reconstruct the missed heads u_i ($i \in M[p]$) at the receiver side, which is necessary in order to update the dictionary.

In our example, if $p = 11$ then $M[11] = \{2\}$ and the only needed difference is $u_2 - u[11,2] = 1$. For the differences in point 2 we have

$$h = \sum_{i \in M[p]} t_i - r - d$$

bits available. If all the differences from point 2 fit into these h bits, then p is a valid choice for the algorithm and all data values x_i with $i \in [1,d] \setminus M[p]$ will be correctly transmitted. If it turns out that there is more than one valid choice for p, then one could choose that p that allows to transmit the maximal number of data values. There might also be scenarios where some of the data streams are more important; then one would give priority to these streams. If no valid index p is found then a full miss will be indicated via the bit sequence 0^r.

Especially for large dictionary sizes, where r is large, h becomes small. However, for a large dictionary (e.g., $r > 6$) one has the option to use more than one dictionary index to indicate a partial miss without increasing the number of misses significantly. If we decide to reserve 2^k dictionary indices (for some $k < r$) to indicate partial misses (which results in the dictionary size $2^r - 1 - 2^k$), k bits can be additionally assigned to h as each of the 2^k reserved indices now refers to a certain part of the dictionary which can be encoded with less bits.

3.5. Grouping of active hypercubes

For the cache-based compression algorithm from Section 3.2 (where all offsets are zero) each active hypercube is maintained individually in terms of its corresponding head tuple in the dictionary, i.e., only in a miss case, the sender and the receiver

synchronously update the dictionary by replacing the least-recently-used head tuple by the missed head tuple. The dynamic version of the algorithm from Section 3.3 uses an offset parameter to center an active hypercube around the transmitted sample to provide a better coverage of the sample's neighborhood.

The results of [6] show, that at the same compression ratio smaller dictionaries with larger hypercubes outperform larger dictionaries with smaller hypercubes, although the latter constellation manages more active values than the former, e.g., in the paper compare the constellation $d = 1$, $r = 3$, and $t = 8$ ($(2^3-1)\cdot 2^8 = 1792$ active values) against $r = 1$ and $t = 10$ (1024 active values). In many cases a newly inserted small active hypercube is not able to cover the next data value, especially if t is small compared to n (e.g., $t = n/2$). We overcome this disadvantage with new strategies for combining multiple adjacent active hypercubes to larger active regions that cover more values around the last transmitted sample. This can be applied to the individual stream compression and to the simultaneous compression of multiple data streams. In the latter case, various possibilities exist how to form a so called *region* of active hypercubes. Such a region is characterized by a subset of active hypercubes $H[p]$, one of which is defined as the *center hypercube*. An active region has to be connected in the following sense: Take the graph, whose vertices are the (d-dimensional) hypercubes from the region, and where two hypercubes are connected by an edge iff they intersect in a d'-dimensional subhypercube for some $0 \leq d' < d$. Then this graph has to be connected. We specify a group of active hypercubes by a set $P \subseteq \{0,1\}^r \setminus \{0^r\}$ of (usually consecutive) dictionary indices, all having the same offset vector (which we call the offset of the region): $\delta[p] = \delta[p']$ for all $p, p \in P$ with $p \neq p'$. This means that if one hypercube is moved (by changing the corresponding offset vector) then all other hypercubes from the region are moved in the same way. Moreover, there is a distinguished $p_c \in P$ such that $H[p_c]$ is the *center hypercube* of the region, and we call $u[p_c]$ the *center head* of the region. The active region corresponding to P is

$$R[P] = \bigcup_{p \in P} H[p].$$

We impose the same restriction on the offset vectors as in Section 3.3, i.e., offset vectors have to belong to $\prod_{i=1}^{d}[-2^{t_i-1}, 2^{t_i-1}-1]$. After every successful transmission (successful in the sense that no miss occurs) we update the heads and the common offset for the region $R[P]$ that covers the current tuple of data values $x = (x_1, \ldots, x_d)$ in such a way that x becomes the center of the center hypercube. Let $x_i = u_i v_i$ where u_i is the head of x_i and v_i is the tail of x_i. Assume that x belongs to the region $R[P]$, i.e., no miss occurs. We then update every head $u[p]$ and offset $\delta[p]$ for $p \in P$ by $u[p] := u[p] + (u_1, \ldots, u_d) - u[p_c]$ and $\delta[p] := (v_1 - 2^{t_1-1}, \ldots, v_d - 2^{t_d-1})$. Some care has to be taken in case a head $u[p]$ is out the allowed range (e.g., contains a negative entry).

One may use only one active region (i.e., all dictionary indices contribute to the region) or several regions that can be moved independently from each other. If the data values are clustered around several areas then one ideally uses one active region per cluster. In a miss case, the least-recently-used region is moved to cover the current tuple of data values.

Table 2: Head combinations for a two-dimensional region of 7 squares

dictionary index p	$u[p,1]$	$u[p,2]$
001	$u_1 - 1$	$u_2 - 1$
010	$u_1 - 1$	u_2
011	u_1	$u_2 - 1$
100	u_1	u_2
101	u_1	$u_2 + 1$
110	$u_1 + 1$	u_2
111	$u_1 + 1$	$u_2 + 1$

In the one-dimensional case a region is formed from a certain number (say k) of adjacent intervals which yields a single interval $I = [u2^t + \delta, u'2^t + \delta - 1]$ where $u, u' \in \{0,1\}^s$, $k = u' - u$ and δ is the common offset for the region. The size of this interval is $k2^t \geq 2^t$. This large interval provides a better coverage of the current data value and (under the locality assumption) leads to a higher probability that the next data value is also covered by I.

The grouping technique shows its full potential for the compression of several correlated data streams. Multiple correlated signals often show a characteristic behavior in the d-dimensional product space in the sense that they are concentrated around a low-dimensional subspace of the overall product space. This fact is utilized in multi-stream compression with grouping and enables us to significantly reduce the number of lost data values. For instance, data values of two physically related signals might rise and fall in a similar way. They will be predominantly covered by regions that cover imaginary slopes in the two-dimensional product space. The algorithm supports predefined fixed d-dimensional regions with a different number of hypercubes for different dictionary sizes. We defined symmetric regions based on one center hypercube, e.g., a square or rectangle in the two-dimensional case, or a cuboid in the three-dimensional case.

Table 2 shows an example for a dictionary that defines a single active two-dimensional region consisting of 7 squares. This region corresponds to the 7 squares in Figure 5 with the black perimeter. Darker plotted squares indicate a higher hit rate with respect to the center of the region. We see that the marked region is a good choice to cover the majority of all values with a rather small dictionary.

Comparing the covered region of values with single-stream compression, one would need to concatenate 3 intervals in each stream to cover a square that encloses the defined region for the two-dimensional stream space. For this, the single-stream compression requires more bits (e.g., $r_1 = r_2 = 2$, $r = r_1 + r_2 = 4$ compared to $r = 3$ for the combined compression). This effect

scales for larger regions, making multi-stream compression outperforming single-stream compression due to its ability to save bits by limiting the value coverage to the special region of interest.

Let us remark that the grouping technique from this section can be combined with the partial miss technique from Section 3.4. In fact, this combination leads to the smallest number of lost data values in our experimental evaluation for the case that manually defined regions are maintained based on a previous signal analysis.

3.6. Automatic grouping of active hypercubes

Forming a suitable region by grouping active hypercubes according to Section 3.5 requires knowledge about the expected behavior of the correlated signals in the product space. This can be gained through an analysis of the hit rate of hypercubes as Figure 5 shows for our test signals. It is obvious that an improper group constellation decreases the algorithm performance drastically. Furthermore, if the signal behavior changes over time, an initially optimal group constellation might not capture the signals well at a later time.

3.6.1. Monitoring the data streams

We introduce a new strategy for our algorithm that automates the grouping of active hypercubes, and thus, supersedes the need for a manual signal analysis and design of group constellations. By synchronously monitoring the hit rate of those hypercubes that cover the samples (hits and misses within a defined subspace of the overall product space) at the sender and receiver side, the group is formed and maintained automatically and adapts to the signals if the characteristics change. For this, the sender and the receiver each maintain two dictionaries of different size whereof which we denote the smaller one *transmission dictionary* and the larger one *observation dictionary*. The former plays the role of the dictionary D that we have used so far in Section 3, where the dictionary index p (an r-bit code) becomes part of the transmitted code word. However, in this new strategy this dictionary does not store the most recently seen heads at $D[p]$ but stores some indices that address entries of the observation dictionary. For clarification, we denote the transmission dictionary by D_{trans} and the observation dictionary by D_{obs} from now on.

The dictionary D_{obs} stores some heads that form a specially designed static region which we call the *observation region*. The hypercubes of this region form a d-dimensional grid with an odd side length in each dimension. Neighboring hypercubes in the grid intersect in a $(d-1)$-dimensional hypercube. For simplicity, we choose the same side length for the grid in each dimension (one could also choose different side lengths). Since this is an odd number, we can define the center hypercube of the grid in the obvious way. The maximum side length of this grid and consequently the number of required entries of the observation dictionary depends on the size of the active region ($n_{trans} = 2^{r_{trans}} - 1$) maintained in the transmission dictionary. Let us call this region the *transmission region*.

In an extreme case, all hypercubes of the transmission region are linked in a one-dimensional chain of length n_{trans}. This defines the maximum side length of the grid forming the observation region in each dimension. For instance, if $d = 2$ and $r_{trans} = 2$, the number of hypercubes of the transmission region is $n_{trans} = 3$ and consequently the maximum number of hypercubes of the observation region is $n_{trans}^d = 9$. In practice, it turned out that a smaller observation region is sufficient. Our experiments showed that the number n_{obs} of hypercubes of the observation region should satisfy $3 \cdot n_{trans} \leq n_{obs} \leq n_{trans}^d$. Then the observation region is a d-dimensional grid of side length $\left\lceil n_{obs}^{1/d} \right\rceil$.

It should be pointed out that the size of the observation dictionary has no influence on the transmitted code word lengths. Nevertheless, it is advantageous to limit its size for several reasons. Our compression algorithms are especially designed to compress correlated signals (see Section 3). Hence, an alignment of many hypercubes in one dimension is very unlikely. Moreover, a limitation can help to prevent overfitting of the transmission region to the signals. This refers to the fact that short-term signal characteristics are captured well but even slightest signal changes lead to more misses. Generally, the algorithm performance decreases if overfitting occurs.

The strategy for the automatic composition of a transmission region has the goal to select $n_{trans} = 2^{r_{trans}} - 1$ indices from the observation dictionary. The hypercubes $H[p]$ (see Equation (1)) that are determined by the selected indices p define the transmission region; it is a subset of the observation region. The transmission dictionary will store the selected indices of the observation dictionary. The observation region behaves as described in Section 3.5, i.e., after every transmission the tuple of input data values x lies in the center hypercube of the observation region. Moreover, the observation dictionary stores the offset vector. For the above selection process we store a sliding window consisting of W (for a fixed constant W) observation dictionary indices. For every new tuple of input data values we determine the observation dictionary index p_{obs} that covers x, add p_{obs} to the sliding window and remove the oldest index from the sliding window. If x does not belong to the observation region (and hence is not covered by an observation dictionary index) then the sliding window is not affected. These operations are performed on the sender and receiver side in a synchronized way. Note that it may happen that x does not belong to the transmission region (in which case a miss occurs) but belongs to the observation region. In this case, sender and receiver modify the sliding window in the same way.

3.6.2. Static transmission dictionary updates

After a predefined period t_1 (e.g., some seconds) the sliding window is analyzed by counting for every observation dictionary index the number of occurrences in the sliding window. The n_{trans} observation dictionary indices $p_1, \ldots, p_{n_{trans}}$ with the highest count are determined. The new transmission region finally is $\bigcup_{i=1}^{n_{trans}} H[p_i]$. Accordingly, the transmission dictionary is updated with the selected indices $p_1, \ldots, p_{n_{trans}}$. In this way, the transmission region reflects the temporal occurrence of data tuples with respect to their precursors. Initially, a predefined

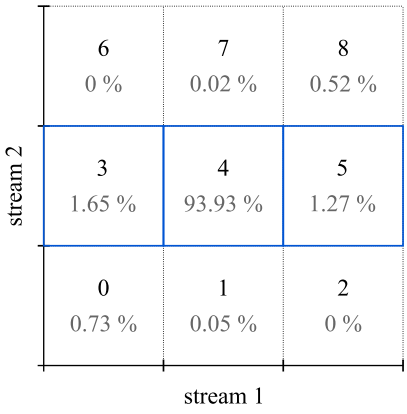

Figure 2: Two-dimensional observation region of 3×3 hypercubes for an active transmission region of 3 hypercubes

transmission region can be implemented at the sender and the receiver side.

A successful sample transmission is possible if there is a transmission dictionary index p_{trans} such that $D_{trans}[p_{trans}]$ (which is an index of the observation dictionary) covers the current tuple of data values x. If such an index p_{trans} does not exist, then we have a miss and indicate this via the reserved entry of the transmission dictionary. Moreover, the observation region is centered around the approximation of x that is obtained from the transmitted tuple of heads (which is known to the receiver as well).

Figure 2 exemplarily illustrates an observation region for the two-dimensional case with $r_{trans} = 2$. Based on the intended size of the transmission region (here $n_{trans} = 2^{r_{trans}} - 1 = 3$), the size of the observation region in each dimension is 3 and every index of the observation dictionary has $\lceil \log_2(3^2) \rceil = 4$ bits (i.e., $2^4 = 16$ dictionary entries of which 9 are actually used). The squares in the figure are named with numbers from $0, \ldots, 8$ and represent the 9 entries (i.e., indices) of the observation dictionary. In the shown constellation $H[4]$ forms the center hypercube. Moreover, the figure exemplarily shows the hit count for all observation dictionary indices as a result of a sliding window evaluation. We see that in 93.93 % of the cases the data tuple to be transmitted was covered by the center hypercube. The active transmission region is consequently formed by the three hypercubes $H[3]$, $H[4]$ and $H[5]$ with the blue perimeter. The entries of the transmission dictionary are then set as follows: $D_{trans}[01] = 3$, $D_{trans}[10] = 4$, and $D_{trans}[11] = 5$. Adding up all shown percentage values reveals that a certain small portion of samples was not captured by the observation region.

The time interval t_1 in which the transmission region gets renewed influences the adaptability to the signals. If a change in the signal behavior occurs, a short time interval offers the possibility to quickly adapt. However, a typical problem then is a low count value for many of the observation dictionary indices in the sliding window. This might prevent a well-founded composition of the transmission region. For instance, if the transmission region is supposed to consist of 31 hypercubes but only 20 different indices show occurrences in the sliding window the strategy fails to compose the transmission region. Increasing the length of the sliding window might help, but a longer sliding window potentially includes more outdated information. In the following we describe a strategy for filling the transmission region in case if there are not enough observation dictionary indices with a non-zero count in the sliding window.

The centers of the hypercubes $H[p]$, where p is an index of the observation dictionary (let us call them observation hypercubes in the following) form a set of points in a d-dimensional space. Each such point then corresponds to a unique observation dictionary index. We declare the center of the center hypercube to be the origin of the above point set. After an analysis of the sliding window, all those points are replicated with the number of occurrences of their corresponding observation dictionary index in the sliding window. In order to determine the main direction of the signal behavior in the d-dimensional observation space, which should be covered by the transmission region, we perform a principal component analysis (PCA; see e.g., [14, Chapter 8]). Roughly speaking, the algorithm calculates a best fitting line (a vector) through our set of points such that the average squared distance from a point to the line is minimized. A next best-fitting line would then have a direction perpendicular to the first. One can repeat this process to obtain an orthogonal set of basis vectors, which are called principal components.

Figure 3 shows an example from our test signals. The transmission region is supposed to be formed by 15 hypercubes ($r = 4$) but only 13 were selected from the sliding window analysis (marked in blue color). In the two-dimensional observation space the first principal component (pc1) highlights the main direction of the signal behavior (we neglect pc2 as only one main signal direction is to be expected). It is a good choice to extend the transmission region in that direction.

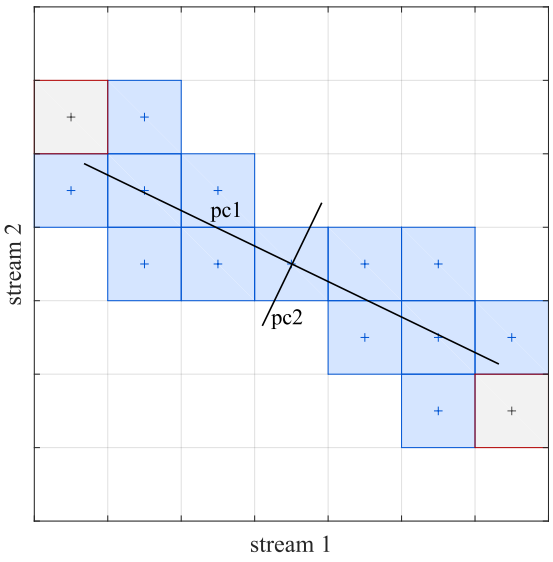

Figure 3: Forming a suitable region in case the number of different observation dictionary indices in the sliding window is insufficient

To determine additional observation hypercubes (or equivalently, the corresponding indices of the observation dictionary)

to be added to the transmission region, for every point the minimal distance to the line (i.e., the first principal component) is calculated. The minimal distance is computed along a line that runs from the point and is perpendicular to the target line. The distance values are sorted in ascending order, i.e., the point closest to the line comes first. Recall that each point corresponds to a unique observation hypercube. We then fill the transmission region with observation hypercubes in the sorted order (where, of course, we only select hypercubes that are not already part of the transmission region) until the desired size is reached. In the example in Figure 3 the transmission region is extended by the two hypercubes drawn in gray color with a red perimeter.

When applying this strategy the shape of the transmission region depends on the size of the observation region. The larger it is, the more hypercubes lie close to the line and the final region becomes wide-stretched. Limiting the observation region to a reasonable size (recall that $3 \cdot n_{trans} \leq n_{obs} \leq n_{trans}^{d}$) prevents this behavior.

Furthermore, we propose the following strategy to allow a fast adaptation in case the signal behavior changes drastically within a short time interval (also consider the case that a fault occurs in the system). We define a second time interval t_2 that is shorter than the fixed renewal period t_1 for the transmission region. Whenever possible (a certain number of different indices is required in the sliding window) the principal components are processed after t_2 time steps. As long as the transmission region properly reflects the signals, the direction of the first basis vector (pc1) of two consecutive t_2-cycles does not vary much. However, an abrupt change of the direction indicates a changing signal behavior such that the formation of a new region can be triggered in between the normal t_1-cycles, which works fine if the sender and the receiver realize the strategy based on the same parameters.

3.6.3. Dynamic transmission dictionary updates

The above strategy for updating the transmission region is based on fixed time intervals (t_1 and t_2). One can make this update process more flexible once a transmission region has been formed with the above strategy. For this we monitor the misses that occur inside the observation region, which occurs if a data tuple is covered by an observation dictionary index, which however, is not stored in the transmission dictionary. These misses frequently occur if the shape of the transmission region does not cover the data tuples well and might indicate a change in the signal behavior. In every such miss case, the sliding window is evaluated (synchronously at the sender and the receiver). We seek the unique observation dictionary index with the lowest count, which is part of the transmission dictionary. Let this be p_1 stored at $D_{trans}[p']$. This count is then compared to the count of the observation dictionary index where the miss occurred, let this be p_2. If p_2 occurs more often in the sliding window than p_1, there is a high chance that upcoming data tuples will be covered by p_2. We therefore replace p_1 in the transmission dictionary by p_2 by setting $D_{trans}[p'] = p_2$. Otherwise, the transmission dictionary remains unaffected.

This dynamic update process of the transmission region offers the potential to immediately and smoothly react to changes in the signal behavior. However, since updates are triggered by miss events and only refer to a single observation dictionary index at one time step, it does not offer the forward-looking character which the strategy based on the principal components offers. A combination of both strategies is possible. The transmission region then gets completely renewed after the period t_1. In between that time interval, the dynamic renewal process adapts the transmission region to the signals if required.

In fact, without any prior analysis of the signal behavior in the d-dimensional product space over time, the strategy of the automatic grouping of active hypercubes outperforms the other schemes due to its ability to automatically adapt to the signals and optimally form new regions.

4. Examples and evaluation

4.1. Simulation model

The recent developments in the fields of automotive and industrial applications demand highly reliable systems, which are able to monitor and diagnose themselves from a large number of sensor data or state variables. In order to guarantee the real-time requirements of these systems, time-triggered architectures are utilized, where the complete communication of data streams is predetermined and scheduled. Fixed compression ratios are needed for this.

Up-to-date fault-diagnosis techniques are based on machine learning, neural networks and artificial intelligence and require the constant processing of large amounts of input data [15]. The complexity of these systems arises, amongst others, due to the spatial decomposition, where sensor data are gathered, merged, and processed at different locations, e.g., processing units or cloud services. It is our goal to show that a combined compression of multiple data streams is able to reduce the size of messages compared to an individual stream compression. This enables a scheduling algorithm to optimize the task allocation and consequently the makespan.

Typical sensor data used for fault-diagnosis are voltage and current measurements at electric components which need to be communicated from their origin at the sensors to the relevant processing units through a distributed network. For the analyses and evaluation of our compression algorithms we simulated data with the hybrid-electric vehicle (HEV) Simulink model [16]. It allows to realistically simulate sensor measurements and state variables at the various included electrical and mechanical components according to a driving cycle input. A more detailed introduction to the model can be found in [17]. The experimental results for the simultaneous two-dimensional data stream compression are based on the voltage and current measurements at the DCDC converter of the Simulink model output of a worldwide harmonized light-duty vehicles test procedure (WLTP)-Class 3 driving cycle simulation [18].

Figure 4 shows the two signals, where the physical quantities are mapped to the corresponding quantization levels (analog-to-digital conversion). The signals are suitable for the evaluation as they offer a challenging signal behavior and can be seen as representatives of signals often to be analyzed in diagnosis

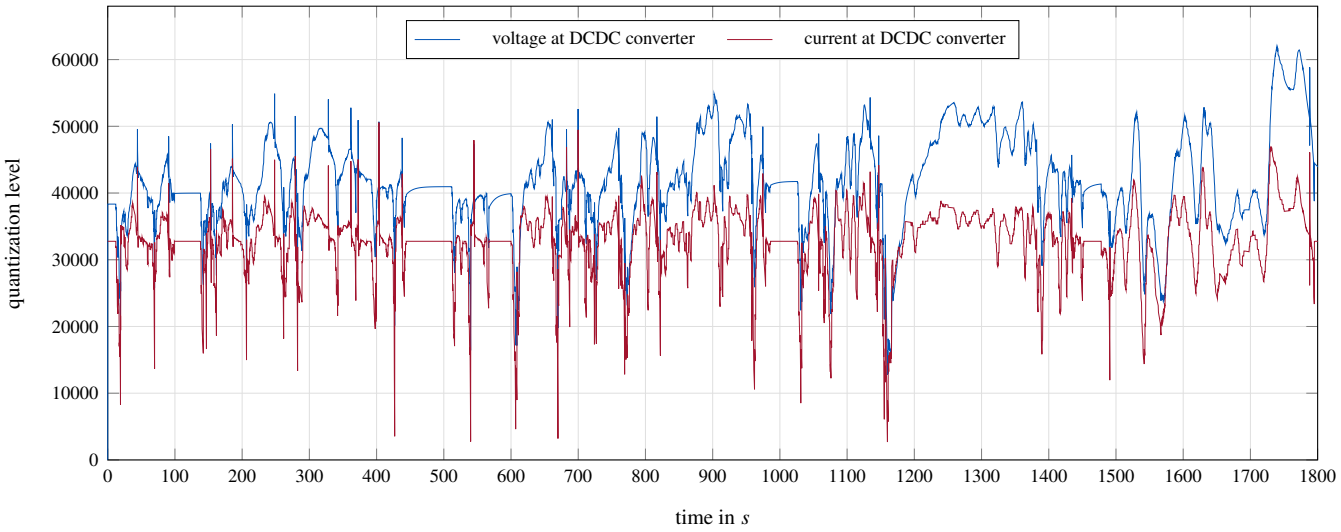

Figure 4: Quantized voltage and current signals at the DCDC converter.

systems. They include a broad coverage of quantization levels, slowly varying as well as rapidly varying signal sequences. Each data set contains $N_s = 180100$ samples, quantized with $n = 16$ bits with a sampling frequency of $100\,Hz$. Moreover, they show a correlation based on their physical relation, i.e., the two signals rise and drop in a similar fashion, often with different offsets. This correlation can be quantified with the Pearson correlation coefficient $\rho \in [-1, 1]$, which is a suitable measure for the similarity of two signals as it indicates a linear correlation. A value of $\rho = 1$ expresses a total positive linear correlation, 0 means no linear correlation, and -1 indicates a total negative linear correlation. Let us denote the two signals with S_c (current) and S_v (voltage). For our data we obtain the Pearson correlation coefficient

$$\rho(S_c, S_v) = \frac{\operatorname{cov}(S_c, S_v)}{\sigma(S_c) \cdot \sigma(S_v)} \approx 0.8129,$$

where cov is the covariance and σ is the standard deviation. The standard deviation of S_c is given by

$$\sigma(S_c) = \sqrt{\frac{1}{N_s} \sum_{i=1}^{N_s} |S_c[i] - \mu(S_c)|^2}$$

($\sigma(S_v)$ is calculated analogously) and the covariance of two signals is defined as

$$\operatorname{cov}(S_c, S_v) = \frac{1}{N_s} \sum_{i=1}^{N_s} (S_c[i] - \mu(S_c)) \cdot (S_v[i] - \mu(S_v)),$$

(in both equations Bessel's correction is not considered) where μ refers to the means of the signals:

$$\mu(S_c) = \frac{1}{N_s} \sum_{i=1}^{N_s} S_c[i] \quad \text{and} \quad \mu(S_v) = \frac{1}{N_s} \sum_{i=1}^{N_s} S_v[i].$$

In complex systems, numerous voltages and currents are measured at different locations since they instantly capture system changes, in contrast to e.g., temperature measurements which typically show a more inert behavior. The implementations can be found on our project website[2].

4.2. Evaluation of the loss rate

This section evaluates the performance of the introduced compression strategies in terms of the achievable loss rate with respect to the compression ratio.

The definition of a suitable shape of an active region in the sense of Section 3.5 significantly influences the compression performance for the simultaneous compression of several data streams. The heat map in Figure 5 shows a region with

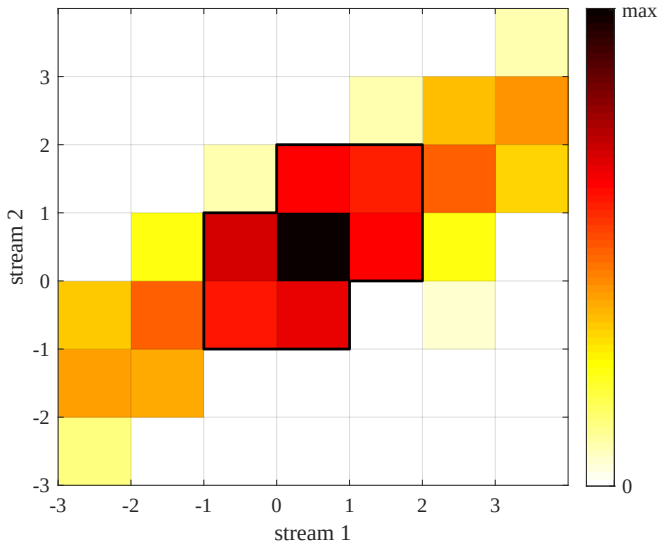

Figure 5: Heat map (logarithmic scale) of the hitting squares around the center. The active region consists of 49 hypercubes (here squares).

[2] https://networked-embedded.de/es/index.php/dakodis.html

$7 \times 7 = 49$ squares and the relative center $u_1 = u_2 = 0$. For the analysis we set $r = 6$ and used only 49 out of the available 63 dictionary entries. According to Section 3.3, the region is always centered around the last transmitted sample. From darker to lighter colors we see that consecutive samples are covered by squares lying on a diagonal of the region. This is expected if two signals rise and fall in a similar fashion. Based on this analysis, a subset of the shown squares can be used for smaller dictionary sizes (see Table 2), or more squares can form larger regions. A white square indicates that no input sample was covered by that square, meaning that this square is dispensable and can be removed from the region without increasing the number of lost data values.

Figure 6 shows the relationship between the loss rate (the total number of lost data values divided by the total number of data values in all streams) and the compression ratio for the combined compression of the two correlated signals with different settings for the parameters r and t. All results are calculated according to the static cache-based algorithm (Section 3.2) extended by the grouping technique from Section 3.5. For all dictionary sizes one active region is maintained. Moreover, we highlight the partial miss strategy (Section 3.4), which reduces the loss rate for all compression ratios, e.g., compare the blue line with the square markers with the red line with the diamond markers. Since $s \leq t$ the lowest possible compression ratio for the two settings is $(r+t_1+t_2)/(s_1+s_2+t_1+t_2) = (1+16)/32 = 0.53125$ and $(1+20)/32 = 0.65625$, respectively. In Figures 6 to 10 the values for r are written to the first three data points of every line; r increases by 1 for every further data point. The lines between the data points are shown for illustration only.

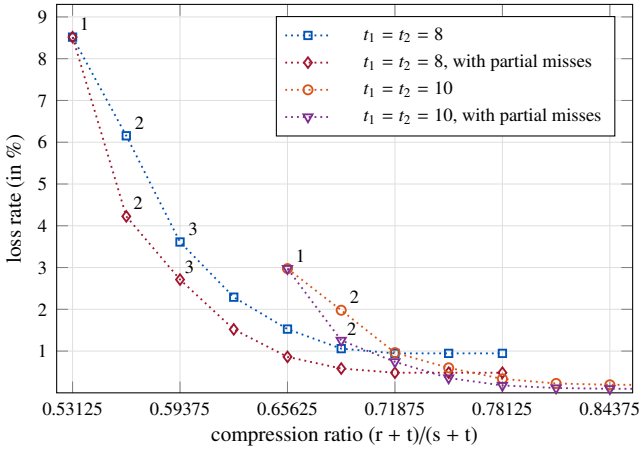

Figure 6: Simultaneous compression of two data streams. Comparison of the cache-based algorithm with its improved version supporting partial misses.

Figure 7 demonstrates the improvements of the simultaneous compression of two data streams when the hypercubes in the dictionary are maintained as one region according to Section 3.5. For the tail lengths $t_1 = t_2 = 8$ (compare the blue line with square markers with the red line with diamond markers) the loss rate drops from 2.2 % to 0.34 % at a compression ratio of 0.625. Moreover, we see that larger dictionaries with smaller hypercubes (here squares) enable to better adjust them to the signal, e.g., at a compression ratio of 0.625, the red line lies below the purple line.

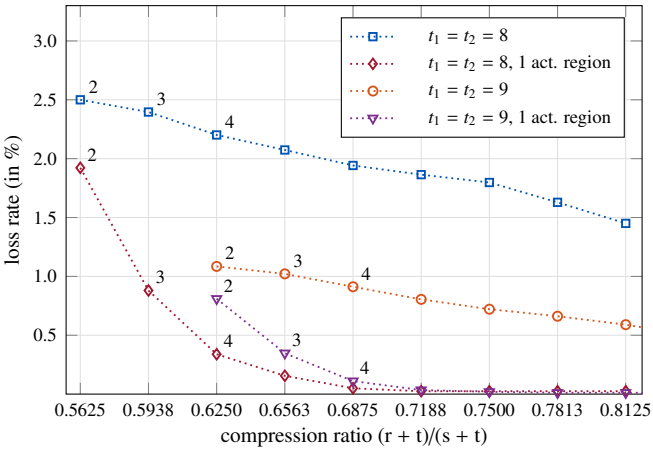

Figure 7: Simultaneous compression of two data streams. Comparison of individually maintained hypercubes and a region of hypercubes.

In Figure 8 we compare the combined compression of the two data streams (according to the dynamic cache-based algorithm with the improvements from Sections 3.4 and 3.5) with the two streams compressed individually (dynamic cache-based algorithm with $d = 1$ and grouping according to Section 3.5), where the loss rate is calculated from the overall number of lost samples in both streams divided by the overall number of transmitted samples. To have comparisons for all compression ratios we performed two simulations for the individual stream compressions, one where the tail sizes are $t_1 = t_2 = 8$ (red line with diamond markers) and another one with $t_1 = 8, t_2 = 9$ (orange line with circles). The results show, that the simultaneous compression of the two data streams outperforms the individual stream compression up to compression ratios of about 0.75, as the blue line with the square markers is the lowest. It is to be expected that this effect advantageously scales with the combination of more correlated streams.

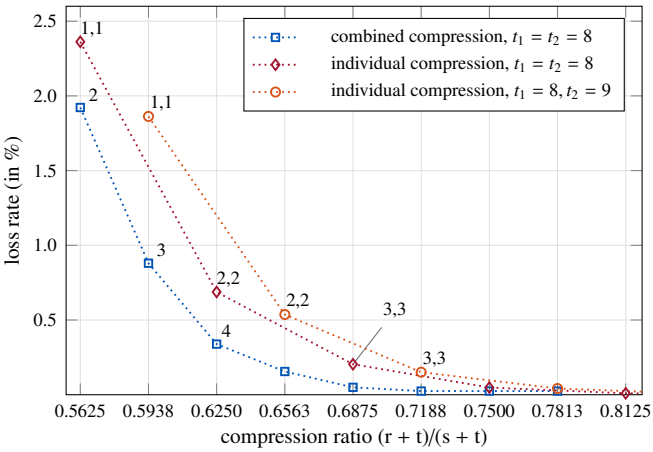

Figure 8: Comparison of the dynamic simultaneous compression of two data streams with the compression of the individual data streams.

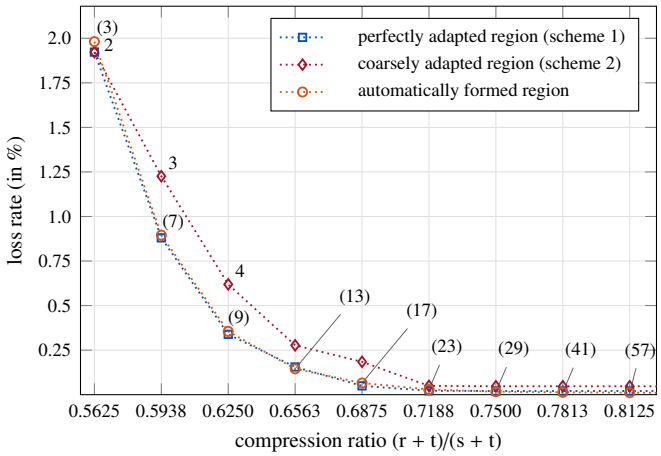

Figure 9: Comparison of different fixed group constellations with the automatic group forming strategy for the voltage and current signals.

Figure 9 compares the loss rate of the data transmission of our test signals for differently shaped fixed regions. In Figures 9 and 10 the values for r are written to the first three data points of the red lines with the diamond markers. They are valid for all lines at the corresponding compression ratio. Moreover, the side lengths of the grid forming the observation region (here $\sqrt{n_{obs}}$) are written in brackets and belong to the orange lines with the circles. For instance, at a compression ratio 0.5938 we have $r = 3$ and the region consists of at most $2^3 - 1 = 7$ squares in the 2-dimensional product space. Recalling Figure 5, the signal analysis reveals the 7 squares with the perimeter to be the best solution to form a region, whereas any other combination of 7 squares would lead to significantly higher loss rates. This procedure is analogously applied for all simulated region sizes. Based on these predefined regions which show an excellent sample coverage, the blue line with the square markers is processed according to the dynamic simultaneous compression (Section 3.3) and shows the lowest loss rate. The red line with the diamond markers stands for a more realistic scenario, where more general forms of regions are applied, which cover the sample's neighborhood better in also other then the predominant directions. There is a benefit in terms of robustness, however, this comes with the cost of a higher loss rate. Still, certain knowledge is included in the region formations, i.e., consecutive samples typically describe a movement on the diagonal from bottom left to top right, and thus, we set the regions to show a symmetry around a line with the slope 1 in the 2-dimensional product space. If no predominant direction of the samples in the product space can be expected within a certain time interval (i.e., no strong dependency between signals), our compression strategies still work, but cannot offer their full potential. The orange line with the circle markers is processed according to the strategy from Section 3.6.2 with the extension from Section 3.6.3 (automatic grouping of active hypercubes supporting dynamic transmission dictionary updates). This combination shows the best performance, as the dynamic transmission dictionary updates explicitly help to reduce the number of lost data values inside the observation region. We conclude that this method shows a loss rate basically as low as the manual strategy based on a prior signal analysis, however, here we do not need any prior knowledge about the expected signal behavior.

The major advantage of the proposed new strategy (Section 3.6) becomes visible in Figure 10. For demonstration purposes we modified one of our testsignals partly to force a different behavior in the 2-dimensional product space, i.e., we inverted the current signal (the red line in Figure 4) for about half of the time such that the signal correlation is reversed. We also simulated a transient fault by making a value of a data stream stuck for a certain time. These signal modifications exemplarily stand for a changing signal behavior over time. It is apparent that the previously introduced fixed regions perform substantially worse in this scenario. For instance, comparing Figures 9 and 10 at a compression ratio of 0.75, the loss rate of the blue curve with the square markers increases from a value close to 0 % to about 0.75 %. The automatically formed regions are able to adapt to the signal changes such that the orange line with the circles shows about the same low loss rate for both scenarios.

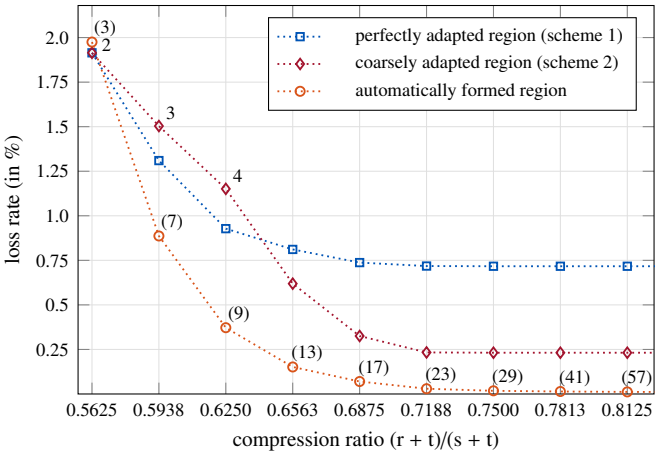

Figure 10: Comparison of different fixed group constellations with the automatic group forming strategy for the *manipulated* voltage and current signals.

4.3. Evaluation of consecutive losses

Besides the loss rate (the accumulated number of lost data values in relation the overall number of values in a certain time interval), other information about loss occurrences is essential. Especially the trend of the number of lost data values calculated from a sliding window might indicate an upcoming change in the signal behavior, being caused by a normal or even faulty system condition.

Another important analysis interprets the number of consecutively lost data values in a data stream. Recall that in a miss case the potential range of the correct data value from the i-th stream has the size 2^{t_i}. For many applications this uncertainty is acceptable if it does not happen too many times in a row. In turn, if an (online) analysis of loss occurrences monitors such incidences, the signal behavior might have potentially changed such that the transmission region became inappropriate. However, it could also just indicate harsh signal conditions. The

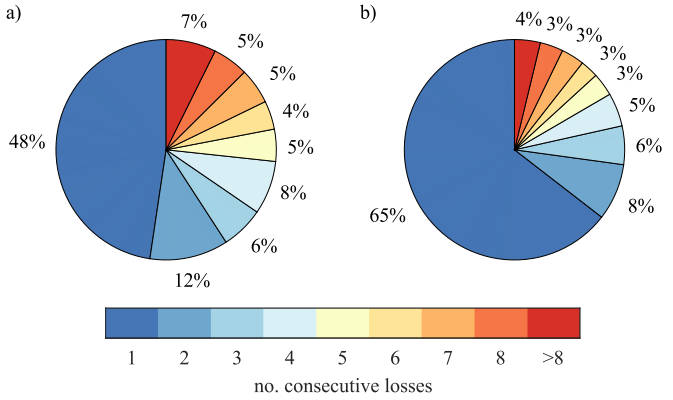

Figure 11: Occurrences of consecutive losses w.r.t. to all loss cases (in percent). Comparison of manually defined regions (chart a) with automatically formed regions (chart b) for the voltage and current signals for $r = 4$.

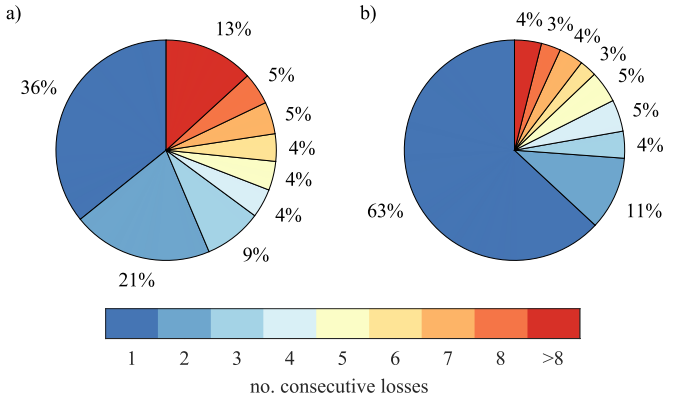

Figure 12: Occurrences of consecutive losses w.r.t. to all loss cases (in percent). Comparison of manually defined regions (chart a) with automatically formed regions (chart b) for the *manipulated* voltage and current signals for $r = 4$.

former case can now be tackled with our proposed new strategy of the automatic grouping of active hypercubes.

The pie charts in Figure 11 visualize the occurrences of consecutive losses with respect to all loss situations and compare the performance of manually defined regions (chart a) with automatically formed regions for $r = 4$ (chart b). Both strategies have about the same overall loss rate (see the blue curve with square markers and the orange curve with circles in Figure 9 at a compression ratio of 0.625). In chart a) (for fixed regions), we see that in 48 % of the loss cases only *one* sample is lost before the region again covers the data (dark blue slice), whereas the corresponding portion for the strategy with automatic grouping of active hypercubes is 65 % (chart b). This shows that the automatic grouping strategy shows a better adaptation to the signals as manually defined fixed regions. The rest of the distribution is also in favor of the automatic grouping strategy, showing that in the majority of the loss cases the region covers the signal again after fewer consecutive lost data values. Looking at the rare cases where more than 8 consecutive data values are lost, it is to be mentioned that a lost data value can be reconstructed with only a slightly lower accuracy, e.g., in our example for stream 1 with $n_1 = 16$ (bits per data value) and $t_1 = 8$, the uncertainty of the correct code word is only half of its potential range of $2^{t_1}/2 = 128$ values of the overall code word space of $2^{n_1} = 65536$ values (if the center value corresponding to the missed head is reconstructed; a detailed analysis of the probability of a loss and the reduction of the uncertainty in this case can be found in [6]). These rare cases might happen if sudden peaks occur in a signal such that the coverage suffers. Yet, the main signal characteristics are captured with our compression algorithm and the loss rate of the analyzed scenario is as low as about 0.35 % (see Figure 9, the square marker and the circle at a compression ratio of 0.625).

Figure 12 is analogously prepared to Figure 11 and shows the analysis for the manipulated voltage and current signals. The pie charts directly reflect the large difference in the loss rate of the two strategies (compare the blue curve with the square markers and the orange curve with circles in Figure 10 at a compression ratio of 0.625). While our new automatic grouping strategy from Section 3.6.2 and Section 3.6.3 performs well in both situations (compare Figure 11b and 12b), the comparison of the pie charts in Figure 11a and 12a reveals a decrease in single loss occurrences as well as an increase of cases with more consecutive losses.

The (online) evaluation of (1) the trend of loss occurrences and (2) the number of consecutive losses can also be included in the triggering process for the renewal of a region. Currently, our trigger interprets a variation of the direction of a continuously processed regression line (in terms of principal components) through the observation region as described in Section 3.6.2. This captures the predominant direction of the signals. Moreover, sudden changes in the signal behavior are detected by monitoring the misses that occur inside the observation region as described in Section 3.6.3. A triggering strategy based on these criteria shows the best performance in our experiments.

The automatic adaptation and monitoring capabilities also form the foundation for an automatic selection of streams that can be advantageously compressed. With synchronized selection schemes, the sender and receiver can agree on some new signals according to predefined evaluation criteria.

5. Future work

In this paper we tackle the challenge for our compression algorithms to automatically adapt to changes in the signal characteristics. The new strategies now support the utilization of the compression algorithms in many different applications, also when only little knowledge about the signals is available. In our current setups, the parameters s, t and r, which determine the number and the size of the hypercubes for each stream as well as the dictionary size and consequently the compression ratio (with a certain loss rate), are fixed. The number of streams d to be included in the combined compression is also fixed. As the trend goes towards open distributed real-time embedded systems (ODRE), where new services (and thus new data streams) can join and leave the system, we intend to support the special requirements in the future by offering possibilities to merge and split data streams online (during the transmission through the network) via adaptive synchronized dictionaries.

As a first step towards this goal we intend to implement improved monitoring and analysis techniques, that simultaneously supervise many data streams, and provide the DAKODIS scheduling algorithms with meaningful information to optimally plan the processing resources and data communications.

In general, our introduced algorithms are mainly designed for the compression of multiple correlated data streams of measured physical quantities, where the data values show a temporal locality. We also want to investigate our algorithms on different kinds of data which show similar characteristics.

6. Conclusion

In this paper we motivate the usage of data compression in distributed online-diagnosis systems. We highlight the fact that diagnostic data streams (e.g., sensor measurements of physical quantities) are often highly correlated and present compression algorithms that allow a simultaneous compression of such data streams utilizing the special signal characteristics. The introduced algorithms support real-time (time-triggered) architectures by guaranteeing a fixed compression ratio. With an example we show that, depending on the parameters, more than one third of the bits (34.3 %) can be saved while still not losing more than 0.2 % of the values (see the fourth data point on the blue curve in Figure 8). In turn, this means that 99.8 % of the values are delivered correctly and without any losses but with a significant reduction of bandwidth demands. The newly introduced compression schemes automatically adapt to changing signal conditions over time and show superior compression results in terms of the lowest loss rate (orange line in Figure 10) and fewer consecutive losses (Figure 12b).

Acknowledgment

This work was supported by the DFG research grants LO748/11-1 and OB384/5-1.

References

[1] R. Isermann, Fault-diagnosis systems: an introduction from fault detection to fault tolerance, Springer Science & Business Media, 2006.

[2] H. Kopetz, Real-time systems: design principles for distributed embedded applications, Springer Science & Business Media, 2011.

[3] M. Paskin, C. Guestrin, J. McFadden, A robust architecture for distributed inference in sensor networks, in: IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, IEEE, 2005, pp. 55–62.

[4] K. Patan, Artificial neural networks for the modelling and fault diagnosis of technical processes, Springer, 2008.

[5] M. Amozegar, K. Khorasani, An ensemble of dynamic neural network identifiers for fault detection and isolation of gas turbine engines, Neural Networks 76 (2016) 106–121.

[6] S. Jo, M. Lohrey, D. Ludwig, S. Meckel, R. Obermaisser, S. Plasger, An architecture for online-diagnosis systems supporting compressed communication, Microprocessors and Microsystems 61 (2018) 242–256.

[7] K. Sayood, Introduction to Data Compression, fourth edition, Morgan Kaufmann, 2012.

[8] S. Majidi, R. Obermaisser, Genetic algorithm for scheduling time-triggered communication networks with data compression, in: 2019 IEEE International Conference on Industrial Cyber Physical Systems (ICPS), IEEE, 2019, pp. 373–379.

[9] S. Meckel, M. Lohrey, S. Jo, R. Obermaisser, S. Plasger, Combined compression of multiple correlated data streams for online-diagnosis systems, in: 2019 22nd Euromicro Conference on Digital System Design (DSD), IEEE, 2019, pp. 166–173.

[10] N. Guan, Techniques for building timing-predictable embedded systems, Springer, 2016.

[11] F. Marcelloni, M. Vecchio, An efficient lossless compression algorithm for tiny nodes of monitoring wireless sensor networks, the computer journal 52 (8) (2009) 969–987.

[12] M. Vecchio, R. Giaffreda, F. Marcelloni, Adaptive lossless entropy compressors for tiny iot devices, IEEE Transactions on Wireless Communications 13 (2) (2014) 1088–1100.

[13] P. L. Dragotti, M. Gastpar, Distributed source coding - theory, algorithms and applications, Academic Press, 2009.

[14] I. T. Jolliffe, Principal components in regression analysis, in: Principal Component Analysis, second edition, Springer, New York, NY, 2002, pp. 167–198.

[15] J. Korbicz, J. M. Koscielny, Z. Kowalczuk, W. Cholewa, Fault diagnosis: models, artificial intelligence, applications, Springer Science & Business Media, 2012.

[16] S. Miller, hybrid-electric vehicle model in simulink, `https://github.com/mathworks/Simscape-HEV-Series-Parallel` (2019, [online; accessed 05-May-2020]).

[17] S. Meckel, T. Schuessler, P. K. Jaisawal, J.-U. Yang, R. Obermaisser, Generation of a diagnosis model for hybrid-electric vehicles using machine learning, Microprocessors and Microsystems (2020) 103071.

[18] United Nations Economic Commission for Europe - World Forum for Harmonization of Vehicle Regulations, Proposal for a new global technical regulation on the worldwide harmonized light vehicles test procedure (wltp), `https://www.unece.org/fileadmin/DAM/trans/doc/2014/wp29/ECE-TRANS-WP29-2014-027e.pdf` (2013, [online; accessed 04-May-2020]).