Compression of Time-Vectors in W7-X Archived Measurements[☆]

J. G. Krom[*,a,b], R. Daher[a], T. Bluhm[a], S. Dumke[a], M. Grahl[a], M. Grün[a], C. Hennig[a], A. Holtz[a], H. Laqua[a], M. Lewerentz[a], H. Riemann[a], A. Spring[a], A. Werner[a], the W7-X Team

[a] *Max Planck Institute for Plasma Physics, Greifswald, Germany*
[b] *European Commission, DG-RTD*

Abstract

WENDELSTEIN 7-X is, as a stellerator, in principle capable of very long operation. The Control and Data-acquisition (CODAC) systems around W7X have, therefore, been designed for continuous operation. One aspect of this design is the recording of the absolute time of every measurement. Whereby for every relevant event, such as every sample-clock tick of every ADC, the time is recorded as a 64 bit integer value.

On the one hand, this approach has several advantages. On the other hand, it generates much more data to process and archive. In extreme cases it can lead to a multiplication of the data amounts.

Compression of these time-stamps-data can be expected to save storage and network resources. However, even though most of these time-stamps have a high degree of predictability, the normal, de-facto standard, compression approaches (ZIP, FLAC, etc., etc.) perform rather poorly on this type of data.

Thus, inspired by well-known audio compression approaches, we developed a specialised loss-less compression algorithm, aimed at fairly constantly incrementing 64 bit long-integers. Experiments on data collected in the past demonstrate very large compression factors, whilst retaining all details in the timing data.

Key words: Data-acquisition, Timing-data, Data-compression

1. Introduction

The WENDELSTEIN 7-X CODAC department runs a uniform and integrated collection of hard- and software systems to measure, compute, store and disseminate signals from W7X and related experiments. These systems have been designed for continuous data-acquisition [1, 2].

One of the unique features of these CODAC systems is the handling of time data.

Older experiments tend to store only descriptive information about the timing of data sources like ADCs. In most cases, that means storing the start-time, in a local time-frame for the experiment, and a sampling frequency.

W7X CODAC decided to measure, record and store a time-stamp for every sampling moment [3, 4]. So, if an ADC runs for 3 seconds, with a sample-clock of 2 kHz, 6 000 time-stamps are recorded, alongside the data from the ADC channels.

These time-stamps are all (1) derived from and synchronised with a central clock, (2) with reference to the same absolute point in time and (3) stored with the same resolution of one nanosecond. Although most acquisition systems have a poorer resolution (10ns or greater), all store these time-stamps as an integer count of nanoseconds since the start of the year 1970, based on the same central clock.

This is a significant quality improvement for the W7X data-acquisition setup over other systems. It allows to correlate measurements from different sources and it reduces the number of assumptions one has to make during analysis. Even if the sample-clock of an ADC is of poor quality, all actual sample-moments are measured instead of assumed, so can be accounted for.

Although this idea to store all these time-stamps is "a good thing", it clearly leads to large amounts of extra data to store. In fairly normal cases, these

[☆]This work has been carried out within the framework of the EUROfusion Consortium and has received funding from the Euratom research and training programme 2014—2018 under grant agreement number 633053. The views and opinions expressed herein do not necessarily reflect those of the European Commission.

[*]Corresponding author: Jon.Krom@ipp.mpg.de

time-stamps can be more than one third of the total data amount. These extra data amounts require extra CPU, network and storage capacity.

In many discussions around W7X, this issue came up and there have been suggestions to reduce the number of recorded time-stamps. One could estimate start and step values from just a few time-stamps and ignore the rest, one could only store every now and then a time-stamp (say, every one-hundred samples), etc.

All these suggestions imply throwing away part of the measured time-stamps and thereby losing all the advantages of this absolute time recording system.

An alternative would be to compress these time-stamps in a loss-free way. That would maintain the advantages of the system, whilst removing the main disadvantage.

It appears that very effective, loss-less compression is possible with a relatively simple algorithm that can be implemented in all relevant environments.

2. General Considerations

Most general-purpose compressors handle their input data as streams of bytes, but ignore thereby information in larger units that could lead to better compression. Especially in the audio field, there exist several loss-free compression algorithms, that use knowledge of the data-type (16, 24, 32 bit integer, or even float) to improve compression ratios.

There are, however (to the current knowledge of the authors) no compression implementations that will attempt such "intra-channel decorrelation" on 64bit (or wider) data streams.

It is possible to adapt the ideas from these audio algorithms to W7X's 64bit time-stamp data. Such an approach is presented here, under the working title of "CTV", for Compressed Time Vector.

This algorithm, as described below, and its support codes could be implemented in any programming language, or indeed even in firm- or hardware. A reference implementation was coded in Java. The choice for Java is mainly to ensure well-defined semantics; for example "`long`" always means a 64bit signed two's complement integer. Furthermore, it is the de-facto standard development language at W7X CoDaC.

3. The CTV algorithm

The CTV method is inspired by the open-source products FLAC, PNG and zlib (or ZIP) [6, 8, 9]. It uses, following FLAC, a multi-step algorithm:

- First, reduce the range of values of the samples (because a list of mainly small values compresses more effectively than a similar list of large values).
- Secondly, compress these reduced values with a general purpose compressor.
- Optionally, pack the data closer together.
- Lastly, pack the compressed data in a "container" structure that makes it easy to differentiate compressed data from normal data.

3.1. First Step: Sample- and Residue Prediction

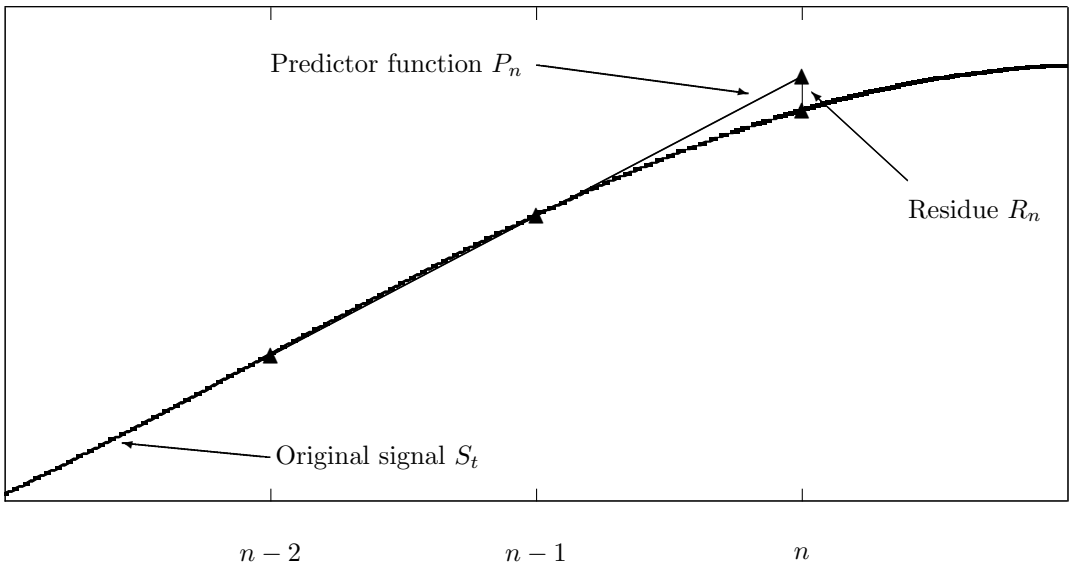
The first step in compressing is trying to remove information in the samples that is already available in earlier samples; or to "de-correlate" data within the channel. A possible way is to predict the value of a sample, based on other samples and then only store the difference between the prediction and the actual sample-value, the "residue". See figure 1.

Note that P_n only depends on S_m for $m < n$. So from a vector of signal samples, a unique vector of residue values can be obtained. From this vector of residue values, the original signal can be fully reconstructed.

The range of values in the residue signal depends on the fit of the predictor to the actual signal. In the case of the W7X time-stamps, a linear prediction function as sketched in figure 1 provides a near-perfect fit, resulting in a residue signal of mostly zeros. A good signal for further compression.

3.2. Second Step: Residue Compression

In principle any general purpose compressor could be used to compress the residue signal. The expected "runs" (or long sequences) of zeros make "Run-length Encoding"[10] an obvious candidate. Hereby are runs of data values (that is, sequences in which the same value occurs in many consecutive elements) stored as a single value V and a count C, rather than as the original run.



Predictor function:	$P_n = S_{n-1} + (S_{n-1} - S_{n-2})$	Assuming: $S_{-2} = S_{-1} = 0$
Residue function:	$R_n = S_n - P_n$	
Reconstructed signal:	$S'_n = R_n + P_n$	

Figure 1: A linear predictor function

3.2.1. Run-length Encoding

Advantages:

- Conceptually easy to understand, implement and test.
- Can be implemented in any environment, including FPGAs; requires only addition (subtraction) and comparison operators on 64bit two's complement integers.
- Very time efficient at run-time, both for compression and decompression.
- Any sample (also the last) can be recovered from the compressed data without the need to fully decompress the signal. Useful in data handling were the first ("`from`") and the last ("`upto`") time-stamp are often referenced.

Disadvantage:

- Sensitive to non-optimal predictors, compression efficiency deteriorates quickly.

3.2.2. Modified Run-length Encoding

Both theoretical considerations and actual measurements at W7X indicate a very high likelihood of two directly subsequent non-zero residue values following each run of several equal residues. (Also following the assumed zero values for indices < 0).

This suggest a useful modification of the pure run-length encoding approach: store these two non-zero residues as they are and then store the run of equal values as a count and value pair.

So, store all the data in a stream of mini-chunks, each of 4 `long`s, see figure 2. The first two longs in a mini-chunk are the residue values, R_q and R_r, without a count (or, an implied count of 1), followed by one pair "count plus value" (C_s, V_s). Any following residue values are encoded in the next mini-chunk.

	$\cdots$	R_q	R_r	C_s	$V_s \cdots$	

Figure 2: A stream of mini-chunks

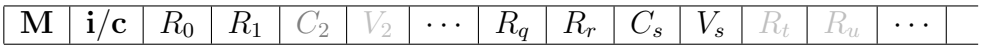


Figure 3: An array of `longs`; a container with a stream of mini-chunks

3.3. Optional Step: Packing

Following (pure or modified) run-length encoding, and perhaps following other compression, it might still be possible to pack the data closer. There will be many small values in `long` words.

In practice, we found this not necessary. Compression is already so good that the remaining timing-data takes only a small fraction of the space required for the associated ADC channel data. We prefer not to incur the added complexity and runtime cost of further packing.

3.4. Final Step: CTV Container

Compressed Time Vectors are packaged into a standard array of 64 bit `long`s, see figure 3. In that way, all software that currently handles time vectors as such arrays can also handle compressed vectors with little or no change.

- The first `long` in this array contains a marker word **M** with the value `0x89435456430d0a1aL`. This is a value that is extremely unlikely to happen in real measurements; it would represent a time-point very far in the future. This value was furthermore inspired by the PNG file marker. The PNG-designers have given their marker some thought [8, p.134].
- A slightly different marker word has been defined (`0x89435456490d0a1aL`) to mark time vectors that were found to be incompressible by an earlier application of the compressor.
- Software could check this very first word and decide, based on that word, if the rest of the array carries real time-stamps or a compressed time vector.
- The second `long` word carries:
  - in the most significant 32 bit, a chunk type identifier **i**, a specifier for the actual compression method, currently `0x4c4d5238L`. Different specifiers should be defined when different encodings, compacting or containers are considered.
  - in the least significant 32 bit, the count **c** of uncompressed `long` words.
- Hereafter follows a stream of mini-chunks, as defined earlier.

4. Assessment

The Java reference implementation of the described algorithm was tested against data obtained from the W7X archive of measurements [5]. The assessment decribed below is based on all data collected on Tuesday[1] 2011-07-26. This data was stored in the archive in "boxes" representing about 1 to 10 seconds of measurements. The time vectors were extracted from these boxes and subsequently compressed, decompressed and checked.

4.1. Reference Assessment

Files carrying an ASCII representation of these time-stamp signals can be compressed with the GZIP program to about 30% of its original size.

Files carrying a binary representation of those same time-stamp signals (which is 0.4 times the size of the ASCII file) can be compressed in that way to about 60% of its original size.

This seems to confirm the generally held believe that loss-less compression gains at best a factor 4.

4.2. Compression Assessment

A first point to note is that all data was correctly decompressed. The CTV mechanism seems indeed to be loss-less.

One can furthermore notice that the data falls into three groups:

- ADC channels that used a low quality, probably a free-running internal clock.

  The 1 565 boxes in this group carried 72 224 000 time-stamps, these compressed into 3 850 568 `long`s, an average compression ratio of 19.
- Data that comes from external sources, PLCs, local control stations, etc..

[1] Although W7X was not yet operational, some components were already being commissioned and produced data.

The 4 019 boxes in this group carried 1 590 731 time-stamps, these compressed into 37 169 `long`s, an average compression ratio of 43.

This ratio is mainly an artefact of reading relatively short time vectors in these boxes, with only a small number of time-stamps per box. Of the 4 019 boxes in this group, 2 946 (73%) compress to the theoretical minimum of 6 `long` words. Using other box sizes will directly affect this compression ratio.

- ADCs that were clocked by a quality, synchronised clock. (It is expected that most of W7X's ADC-clocks will be of this type.)

  These 697 boxes carried 16 350 000 time-stamps, which compressed into 4 242 `long` words, an average compression ratio of 3 854.

4.3. Performance Assessment

Compression speed of this CTV algorithm has been observed around 10 Msample/s, with a not specifically tuned Java implementation, on a fairly standard office PC.

4.4. Worst-case Considerations

For any known compression algorithm, there exist "pathological" input data that, when processed by that algorithm, will actually produce an array that requires more space than the original vector. It is useful to consider such "pathological" data-sets for the described system.

The worst case is an original time vector of one element. This will "compress" into three `long` words.

In fact, any original vector with a length < 6 will "compress" into a longer array. A vector with a length $= 6$ will at best return a data-set of the same length.

Longer original vectors could also cause expansion. This could happen when every residue value is different from its immediate neighbours, so when a fairly random distributed set of time-stamps is recorded. In such cases, the worst-case "compressed" array would have a length of $2 + N \times 4/3$ words.

A CTV compressor can, and should, recognise such expanding data-sets during the compression process. It should return, at that point, a copy of the input array, with a special marker word prepended. If this "compressed" array were to be used again as input to a compressor, it can quickly detect that this data-set was already found to be incompressible.

5. Summary

- A loss-less compression scheme, named CTV (Compressed Time Vector) has been presented.
- This compression scheme is specifically adjusted to 64 bit time-stamps as used by W7X.
- This compression uses "Linear-prediction" and "modified run-length encoding" to significantly reduce the data amounts required to handle, transport and store such time-stamps.
- Typical W7X time-stamp data compresses very well using this scheme.
- All detailed information in the time-stamps is unaffected by this compression.
- This CTV approach is being rolled-out in the W7X archive and related software.

References

[1] A. Werner, *et al.*, Cutting edge concepts for control and data acquisition for Wendelstein 7-X. 10.1109/SOFE.2013.6635430

[2] T. Bluhm, *et al.*, Wendelstein 7-X's CoDaStation: A modular application for scientific data acquisition, Fusion Engineering and Design, 89 (2014), 658-662

[3] J. Schacht, *et al.*, A trigger-time-event system for the W7-X experiment, Fusion Engineering and Design, 60 (2002) 373–379

[4] J. Schacht, *et al.*, The Trigger- Time-Event-System for Wendelstein 7-X: Overview and first Operational Experiences, 20th Real Time Conference

[5] C. Hennig, *et al.*, ArchiveDB—Scientific and technical data archive for Wendelstein 7-X, Fusion Engineering and Design, 112 (2016) 984–990

[6] FLAC – Free, Loss-less Audio Codec, `http://flac.sourceforge.net/`

[7] "Loss-less Compression of Digital Audio (audioPaK)", Mat Hans, Ronald W. Schafer, HP Laboratories Palo Alto, 1999

[8] "PNG, The Definitive Guide", Greg Roelofs, O'Reily. PNG – Portable Network Graphics, ISO/IEC 15948,

[9] zlib – The Compression Library behind PNG, GZIP, java.zip, etc., `http://zlib.net/`

[10] Run-length encoding. E.g.: "Algorithms in C", R. Sedgewick, ISBN 0-201-51425-7, page 320. Or: `http://en.wikipedia.org/wiki/Run-length_encoding`