Reducing the bandwidth and storage requirements for remotely deployed RF probes

LUNA PESHKOV, University of Twente, The Netherlands

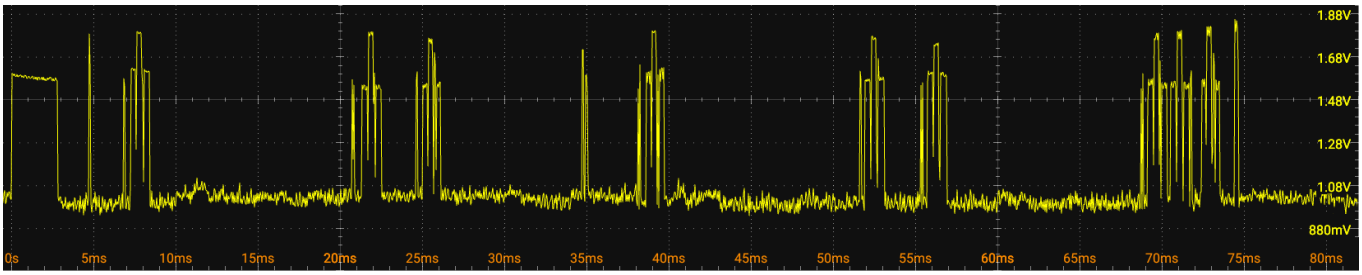

Fig. 1. 80ms of data captured from the RF probe

RF probes produce a large amount of data due to the high sampling rate required to accurately read the data produced by the probe and small IoT devices may not have enough bandwidth available to transmit all that data. Compression algorithms can be used to reduce the size of the data before it is sent to the server. This paper explores using AZTEC and Sprintz to compress the RF probe data and analyses their effectiveness on real data by using metrics line the compression ratio, speed, and lossiness.

Additional Key Words and Phrases: IoT, embedded, RF probe, compression, sensors, bandwidth

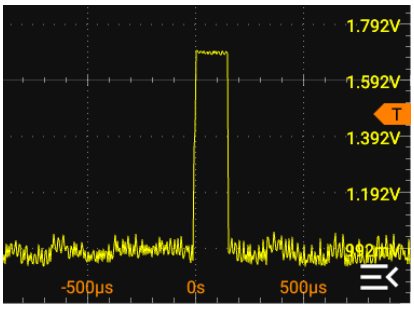

Fig. 2. An example of a single pulse captured from the RF probe

1 INTRODUCTION

Electromagnetic Interference (EMI) is often a result of power electronics [6], wireless devices like wireless access points or mobile phones, and natural sources [15]. EMI can pose a serious problem to electronic devices, which is especially problematic in critical environments like hospitals where radio waves coming from within or outside the hospital [11] can cause medical devices to malfunction [24] which can put human lives at risk.

To combat the potential negative effects of EMI on sensitive equipment, small RF probes can be deployed across a facility to measure the strength of the radio signals present. Areas with high signal levels can then be further investigated to find the source of those signals and determine how much effect those sources have on the equipment around them.

These RF probes deployed can be standalone devices built only for this purpose or can also be a part of some other device like a power quality meter [10] with the probe connected to it externally to be able to correlate the power quality with EMI.

2 PROBLEM STATEMENT

RF probes output a DC voltage which needs to be digitised before it can be sent to the server. Most modern microcontrollers have analog-to-digital converters (ADCs) integrated into them for this purpose. Because of how short electromagnetic pulses can be, the ADC needs to sample the output of the RF probe very often.

Take a look at the 160 µs wide pulse in fig. 2. If we were sampling the signal at let's say 1 kHz, it would have been possible for us to completely miss the peak if we have only took samples at −500 µs and at 500 µs . Let's say we want to capture 80 samples inside the pulse in fig. 2 to have an idea of how long the pulse is, we would need a sampling rate of 500 kHz:

$$\frac{1}{160\,\mu s\ /\ 80} = 500\,\text{kHz}. \tag{1}$$

As a result of the high sampling rate required for this application, a large amount of data needs to be sent to the server or stored

TScIT 41, July 5, 2024, Enschede, The Netherlands

in the probe's internal storage if the connection to the server is unavailable.

Not a lot of bandwidth may be available between the sensor and the server, which will limit the amount of data that can be sent between them.

The bandwidth limitations are also apparent on the server side when multiple probes are sending data to it simultaneously. At the sampling rate of 500 kHz assuming that ADC readings are 16 bits each, 125 sensors can fully saturate a 1 Gbit s^{-1} link:

$$\frac{1\,\text{Gbit}\,\text{s}^{-1}}{500\,\text{kHz} \times 16\,\text{bit}} = 125\ \text{sensors}, \tag{2}$$

and this is without considering any overhead induced by the transmission protocol or retransmitting lost packets.

Data compression can be used to reduce the size of the data sent to the server to overcome the bandwidth limitations. This paper will analyse the feasibility and performance of compression methods.

3 RESEARCH QUESTIONS

In this paper the following research questions are going to be discussed:

RQ1: Which methods can be used to reduce the size of the data obtained from the RF probe before it is transmitted?

RQ2: To what extent can lossy compression methods be used to compress the RF probe data while keeping it usable?

4 RELATED WORK

Not a lot of research has been done specifically on compressing RF probe data, so this paper leverages methods from other fields in addition to sensor-agnostic approaches for compressing time series data.

4.1 General-Purpose Compression Algorithms

LZ4 [5] and Snappy [8] are some of the fastest general-purpose compression algorithms according to [17].

While being great for data like text files, they offer a lower compression ratio for time-series data when compared to other lossless compression algorithms like Sprintz found in section 4.3.1.

4.2 Audio Compression Algorithms

FLAC [23] is a lossless compression algorithm that uses Linear Predictive Coding (LPC) which predicts the value of the current sample based on the values of previous samples and it could potentially work for compressing the RF probe data, but it also does not offer a great compression ratio.

Lossy audio compression algorithms such as MP3 [12] or AAC [13] rely on a psychoacoustic model [4] to compress audio. They cut off frequencies outside the range of human hearing in the process of compressing audio, which is not something that is desirable for compressing RF probe data.

4.3 Algorithms Chosen For This Paper

4.3.1 Sprintz. The lossless compression algorithms that stands out for this application is *Sprintz* [3], which offers a way to compress data with a higher compression ratio than general-purpose compression algorithms like *LZ4* and *Snappy*, while also offering great efficiency which makes is possible to run it on embedded devices.

The high compression ratio is achieved through the use of techniques like delta coding and bit packing.

Compressing data using Sprintz comes down to the following sequence of steps:

(1) Forecasting using Delta Coding
(2) ZigZag Encoding
(3) Bit Packing
(4) Run-length Encoding

Delta coding works by only storing the differences (deltas) between current and the previous value instead of storing full values. Equivalently, it can also be said that delta coding predicts the next sample to be equivalent to the current sample and then stores the error between the predicted and the actual value. This can greatly reduce the amount of bits used to store a stream of values if those values do not change very often.

Since deltas are signed values, in order to efficiently bitpack them we need to convert them to a representation which will keep the leading zeros for small values as if the values were unsigned. One such technique is known as ZigZag encoding [16]. It works by representing positive integers as even numbers and negative integers as odd numbers which preserves the leading zeros for small numbers and allows us to use bit packing to reduce their size.

Bit packing involves packing each value as close as possible based on the amount of bits that is required to represent the highest number h in a sequence of values.

The number of bits ($nbits$) in such a sequence can be expressed by:

$$nbits = \left\lceil \frac{ln(h+1)}{ln(2)} \right\rceil \tag{3}$$

.

For example if we have the following 8-bit values that we want to bitpack **00011000** and **00001101**, we could represent each value in:

$$\left\lceil \frac{ln(24+1)}{ln(2)} \right\rceil = 5\ \text{bits} \tag{4}$$

since the largest value in the sequence is **00011000 (24)**.

We can then store these values right next to each other by truncating both values to 5 bits and by adding a header to tell the decoder that each value is 5 bits: **101 01101 11000** which allows us to save some space.

Sprintz uses run-length encoding to save space when an entire block of values consists of errors which are equal to 0. It works by only writing the number of blocks full of zeroes instead of writing those blocks as usual.

4.3.2 AZTEC. This is a lossy compression technique originally proposed in [7] for compressing ECG data. It represents data by a sequence of lines and slopes.

A line is formed when there is a sequence of data points where the difference between the maximum and the minimum value does not exceed a certain threshold T. Let's say that we have set the threshold T to 7. In fig. 3 we can see that points 0-8 fit under that threshold

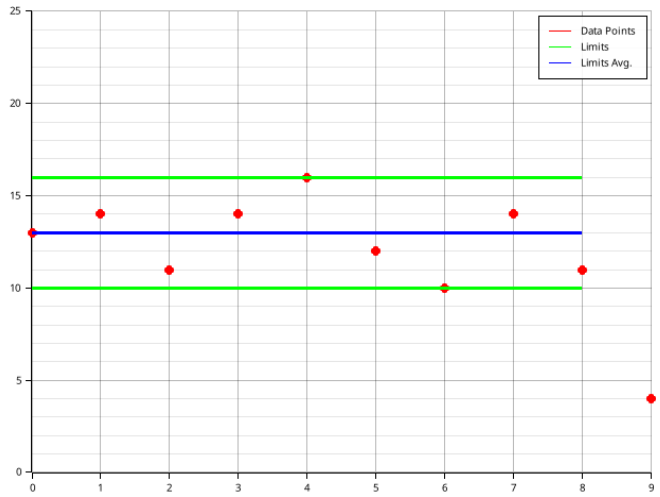

Fig. 3. AZTEC line segment

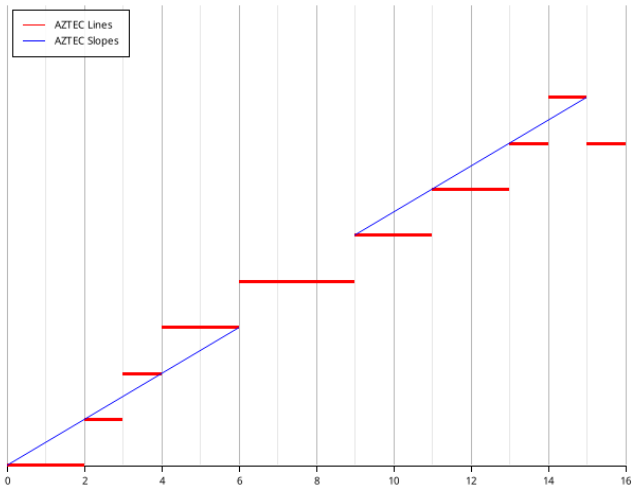

Fig. 4. AZTEC slopes

because $16 - 10 <= 7$, but point 9 will set the new minimum value to 4 and $16 - 4 > 7$ so a new line will be formed starting from point 9 and points 0-8 will be saved as a line with the duration of 9 and the value of 13.

In case there is a fast-changing signal, the data will be represented by a ladder formed by horizontal lines because T is exceeded on almost every sample. This is not very efficient because we will have to store each individual line. This is where the slopes are used. If there is a sequence of lines where the length of each line is lower than a certain threshold L and there is no change in direction between the lines, it means that these lines can be represented by a slope.

With threshold L set to 3, fig. 4 shows that two slopes have been formed by lines whose length is < 3. The line in the middle is not included because it is longer than 3 samples in length and the last line is not included because the line is going into the opposite direction from the slope.

During decompression the value of a line is repeated for the amount of times equal to the duration of the line and linear interpolation is used in order to turn the starting and ending point of a slope as well as its duration into the amount of data points equivalent to the duration of the slope.

While *AZTEC* did not achieve acceptable fidelity for ECG signals [14], it could potentially achieve a great compression ratio without losing important information like the amplitude or the duration of each peak. This is because of how the RF probe data is made up of plateaus and slopes which is exactly what this compression technique uses to describe the data.

5 PROPOSED METHODOLOGY

Sprintz and AZTEC compression algorithms from section 4 will be implemented on a microcontroller with an RF probe connected to one of the ADC channels.

The effectiveness of each compression algorithm will then be analysed based on how much data is sent by the sensor before and after compression and how much compression affects the original signal.

5.1 Modified AZTEC

A slightly modified version of the AZTEC compression algorithm is proposed and will be implemented as well. This modification is based on the fact that we do not necessarily need to know the exact time down to a sample of when an electromagnetic event has occurred. Even only having an approximate time can still allow the engineer looking at the data to know what kind of equipment has been used at that time to cause that pulse. Making this assumption allows us to compress the data even further by only keeping the sections which are above the noise floor.

This way some temporal resolution will be lost because the unique time when a specific peak starts and ends within a chunk can no longer be determined, but we can still determine the point in time at which the peak has occurred within:

$$\frac{1}{200\,\text{kHz}} \times 4096 = 20.48\,\text{ms} \tag{5}$$

if we use a sampling frequency of 200 kHz and compress measurements in chunks of 4096 samples.

5.2 Hardware Details

The probe used is based on an AD8313 [1] logarithmic amplifier connected to a monopole antenna. This RF probe produces a DC voltage (V) based on the input amplitude (dBm).

Modern IoT device range from having small microcontrollers such as ATmega328P running at 16MHz and with 2KB of RAM to single board computers like Raspberry Pi [21] which has a 2.4GHz quad-core Cortex-A76 CPU with 4-8GB of RAM [22]. In the process of writing this paper the author chose to use ESP32 which is powered by a dual-core Xtensa CPU running at 240MHz and has 520KB of RAM [9] as a point of reference for the performance of common IoT

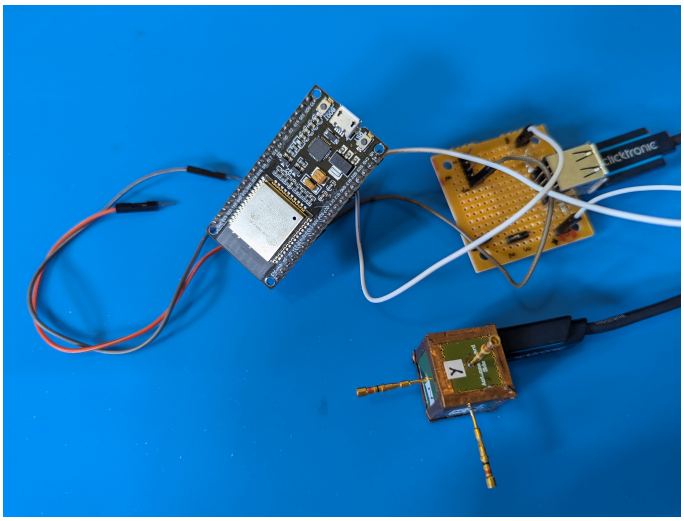

Fig. 5. Experimental Setup

devices because it has been used in many commercially successful IoT devices from companies like Shelly, Xiaomi, and Sonoff [18].

The ADC in the ESP32 is capable of sampling rates of up to 2 MHz [9] which should be more than enough based on the observations made in section 2.

In fig. 5 you can see the RF probe connected to the ESP32. The probe is powered by 5V and its output is connected to GPIO32 on the ESP32 which corresponds to channel ADC1 channel 4.

The wires on the left were used for the purpose of measuring how long it takes for the compression algorithm to run. A pin is toggled high before the compression starts and then toggled low right after the data has been compressed. Toggling a pin is really fast because it is done in a single CPU instruction. The width of the pulse was then measured by an oscilloscope.

6 EXPECTED RESULTS

It is expected that the compressed data will be smaller than the original data, reducing the bandwidth required to send it to the server. It is also expected that the lossy compression methods will have minimal impact on the usability of the signal.

7 IMPLEMENTATION DETAILS

7.1 Sprintz

The implementation of the Sprintz compression algorithm used is based on the code published by one of the authors of the original paper in a GitHub repository [2].

In order to make the code compile for the Xtensa architecture, the logic related to decompression has been removed because it relies on AVX instructions which are not present on microcontrollers.

Additionally, emulation for the `_lzcnt_u32` and `_pext_u64` x86 intrinsics was implemented and can be found in appendix A in order to make the compression code work on an ESP32. The `lzcnt_u32` instruction counts the number of leading zero bits. The `pext_u64` instruction takes a value and the corresponding mask and extracts

Description	1 - slope 0 - line	Len(samples)	ADC MSB	ADC LSB
Bits	7	6-0	7-0	7-0
Byte	1		2	3

Table 1. The Layout Of One AZTEC Packet

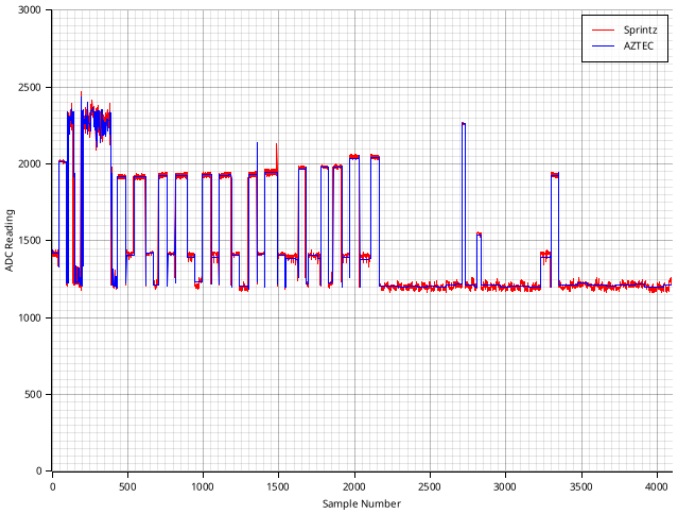

Fig. 6. Comparison of the losslessly compressed data to the data compressed by AZTEC

the bits that are set in the mask from the value and places them in the lower bits of the destination register.

The original code had a memory leak where it allocated memory for a buffer but didn't free it, which is especially critical on a microcontroller because the heap will quickly fill up. This has been fixed in [20].

7.2 AZTEC

The AZTEC compression was implemented from scratch and the implementation can be found here: [19]. The first byte was used to indicate whether or not the current section is a line or a slope as well as to indicate its length. The next two bytes were used to store the value read from the ADC in big-endian order. The exact packet layout can be found in table 1.

In order to select an appropriate threshold, we can look at peak-to-peak value of the noise floor, which appears to be around 100. By setting the AZTEC threshold to 100, we make sure that the noise floor is compressed to a line that is as long as possible and any peaks above the noise floor will still be preserved.

In fig. 6 you can see the same data being compressed by AZTEC as well as Sprintz.

7.3 Modified AZTEC

The compression algorithm could easily be modified to discard the lines and slopes with values which are less than a certain threshold.

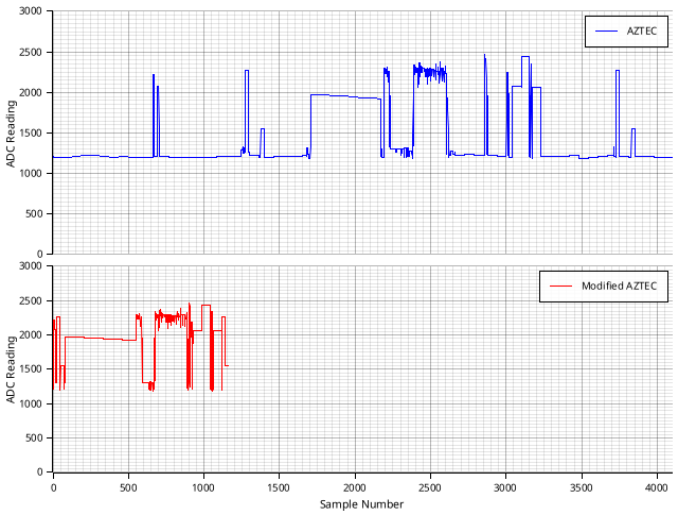

Fig. 7. Modified AZTEC algorithm

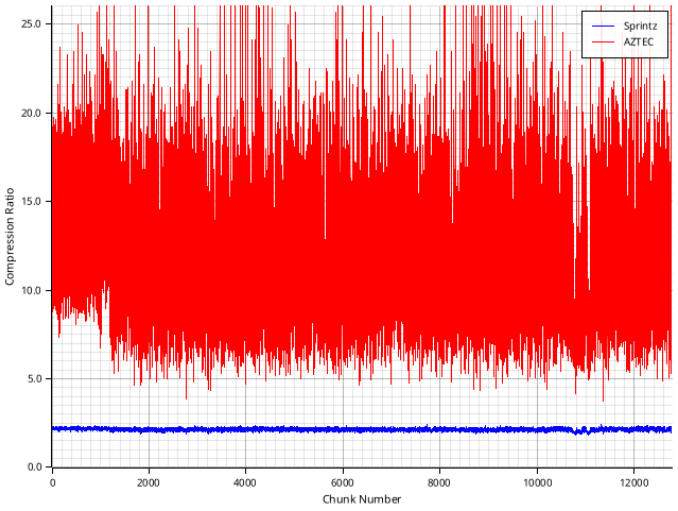

Fig. 8. Variation of the compression ratio with each chunk

Algorithm	Mean	Std. Dev.	Min	Max	RMSE
Sprintz	2.15	0.07	1.85	2.41	0.00
AZTEC	11.66	3.88	3.72	33.99	86.54
Modified AZTEC	21.62	15.00	4.19	372.37	N/A

Table 2. Table showing the compression ratio results for each algorithm

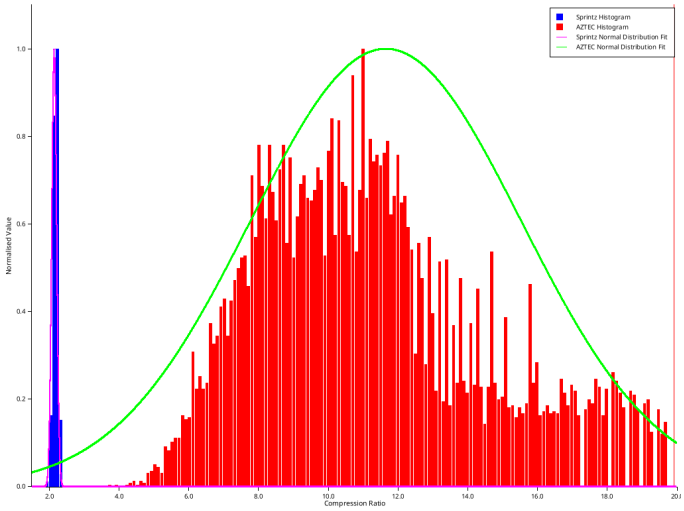

Fig. 9. Compression ratio distribution (normalised)

This threshold is based on the current level of the noise floor. See section 5.1 for the motivation behind this.

In fig. 7 you can see the same data being compressed by the regular and modified version of AZTEC with the discard threshold set to 1300. This discards all lines and slopes with values lower or equal to 1300. You can see that the noise floor has been removed which makes the data even smaller.

8 RESULTS

The comparison between algorithms was performed by measuring their compression ratio, speed and lossines. These values were obtained from recording 4 min 21 sec of data at a 12 bit resolution, with 12 dB attenuation, and the sampling rate of 200 kHz using an RF probe connected to one of the ADC channels on the ESP32 and then using each of the compression algorithms to compress the same data in chunks of 4096 samples.

8.1 Compression Ratio

Compression ratio is defined by:

$$\text{Compression Ratio} = \frac{\text{Uncompressed Size}}{\text{Compressed Size}}. \tag{6}$$

The higher the compression ratio, the better the compression algorithm is able to compress the data. The compression ratio results can be found in table 2 and will be further summarised and discussed below.

Out of the compression methods tested in this paper, AZTEC managed to achieve the mean compression ratio of 11.66 followed by Sprintz at 2.15.

Using the modified version of AZTEC gave the compression ratio of 21.62.

In fig. 8 we can see how the compression ratio varies over each chunk. *AZTEC* has a large standard deviation of 3.88, but the compression ratio never drops below 3.72. *Sprintz* on the other hand shows very consistent compression ratio with a standard deviation of 0.07.

The compression ratio of Sprintz did not have a lot of variation. Sprintz uses run-length encoding in order to store consecutive blocks full of zero errors which would significantly increase the compression ratio if there were a lot of these blocks. See section 4.3.1

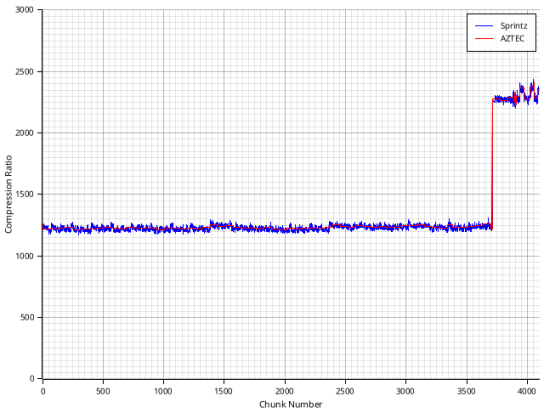

Fig. 10. The chunk which resulted in the highest compression ratio for AZTEC

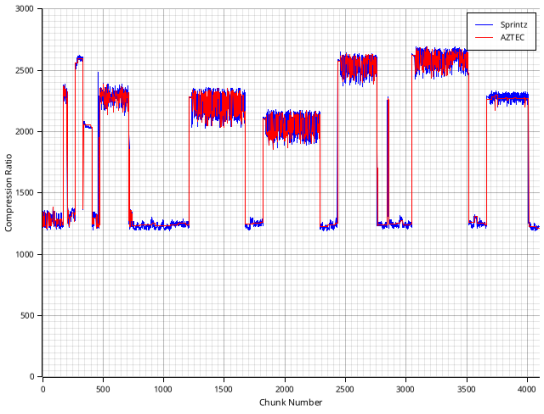

Fig. 11. The chunk which resulted in the lowest compression ratio for AZTEC

for more information. RF probe data has a lot of noise and therefore it is really difficult to get a packet full of errors equal to 0. Because of this Sprintz has to store each individual packet which results in a pretty consistent compression ratio without any significant peaks of increased compression ratio.

As to why the compression ratio of AZTEC varies so much, let us look at the chunks with the highest and with the lowest compression ratios.

As expected, the chunk with the highest compression ratio shown in fig. 10 is almost entirely made up of the noise floor which can be represented as a series of very long lines and can therefore be compressed very well.

The chunk with the lowest compression ratio seen in fig. 11 has peaks with large variation on the top. This variation is higher than the threshold which causes a lot of extra lines and slopes to be created which decreases the compression ratio.

8.2 Speed

The performance of each compression algorithm is very important for this application because the data needs to be compressed before the next batch of samples arrives from the ADC. AZTEC ran the fastest, taking only 2.95 ms on average to compress 4096 samples on the ESP32, while Sprintz took 18.26 ms to compress the same amount of samples.

8.3 Lossiness

As shown in fig. 6, the RF probe data is still usable (magnitude and duration can be determined) after being compressed and then decompressed by the AZTEC compression algorithm. Compressing the data with AZTEC resulted in an average Root Mean Square Error (RMSE) of 86.54 which is less than the threshold of 100 that we set to be used by the compression algorithm. The amplitude and the start and stop times of the areas above the noise floor are well-preserved. RMSE is not applicable for the modified version of AZTEC because the amount of samples on the output are not equal to the amount of samples on the input.

Data compressed by Sprintz is inherently as usable as the original data. It has a RMSE of 0 because it is a lossless compression algorithm which means that the data can be decompressed from its compressed state to exactly the same data that was initially compressed.

9 FUTURE IMPROVEMENTS

Instead of using a fixed threshold to discard the noise floor in the modified AZTEC compression algorithm, a noise floor estimation technique could be employed to determine the noise floor for each chunk individually in case it changes based on environmental factors.

The probe used in this paper has 3 channels for X, Y, and Z, but only one of them was used to capture and compress the data. Reading and compressing more than one channel of the probe can implemented in the future.

This paper has only evaluated two promising compression algorithms. In the future more compression algorithms can be tested to get a better idea of which ones work for this purpose and which ones do not.

10 CONCLUSION

Both Sprintz and AZTEC were able to run on the ESP32 to compress the RF probe data data sampled at 200 kHz in real time and have achieved respectable compression ratios of 2.15 and 11.66 respectively. Taking a more lossy approach allowed us to almost double the compression ratio of AZTEC to 21.62 by discarding the noise floor.

AZTEC took 2.95 ms and Sprintz took 18.26 ms to compress 4096 samples of data on the ESP32. This gives the per sample compression times of: 0.72 µs and 4.46 µs which allows for compressing the data at the maximum sampling rates of 1.39 MHz for AZTEC and 224.32 kHz for Sprintz. This puts AZTEC at an advantage for compressing data captured at higher sampling rates.

The lossy compression method tested (AZTEC) did not have a significant impact on the usability of the RF probe data because the original data closely resembled the data that went through compression which is shown by a RMSE of 86.54. Lossy compression

methods are preferred for this kind of data due to their high compression ratio to the extent where the peaks and their durations are preserved, which is the case when using AZTEC.

REFERENCES

[1] Analog Devices 2023. *AD8313: 0.1 GHz-2.5 GHz, 70 dB Logarithmic Detector/Controller Data Sheet.* Analog Devices. https://www.analog.com/media/en/technical-documentation/data-sheets/ad8313.pdf Rev. F.

[2] Davis Blalock. 2021. sprintz. https://github.com/dblalock/sprintz

[3] Davis Blalock, Samuel Madden, and John Guttag. 2018. Sprintz: Time Series Compression for the Internet of Things. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 2, 3, Article 93 (sep 2018), 23 pages. https://doi.org/10.1145/3264903

[4] Karlheinz Brandenburg. 1999. MP3 and AAC Explained. *Journal of the Audio Engineering Society* 17-009 (august 1999).

[5] Yann Collet. 2011. LZ4. https://github.com/lz4/lz4

[6] F. Costa and D. Magnon. 2005. Graphical analysis of the spectra of EMI sources in power electronics. *IEEE Transactions on Power Electronics* 20, 6 (2005), 1491–1498. https://doi.org/10.1109/TPEL.2005.857564

[7] J. R. Cox, F. M. Nolle, H. A. Fozzard, and G. C. Oliver. 1968. AZTEC, a Preprocessing Program for Real-Time ECG Rhythm Analysis. *IEEE Transactions on Biomedical Engineering* BME-15, 2 (1968), 128–129. https://doi.org/10.1109/TBME.1968.4502549

[8] Jeff Dean, Sanjay Ghemawat, and Steinar H. Gunderson. 2011. Snappy. https://google.github.io/snappy/

[9] Espressif Systems 2023. *ESP32-WROOM-32E Datasheet.* Espressif Systems. https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32e_esp32-wroom-32ue_datasheet_en.pdf Version 1.6.

[10] Roelof Grootjans and Niek Moonen. 2023. Design of Cost-Effective Power Quality and EMI Sensor for Multinode Network. *IEEE Letters on Electromagnetic Compatibility Practice and Applications* 5, 4 (2023), 131–136. https://doi.org/10.1109/LEMCPA.2023.3294129

[11] Eisuke Hanada, Kenji Kodama, Kyoko Takano, Yoshiaki Watanabe, and Yoshiaki Nose. 2001. Possible Electromagnetic Interference with Electronic Medical Equipment by Radio Waves Coming from Outside the Hospital. *Journal of Medical Systems* 25, 4 (01 Aug 2001), 257–267. https://doi.org/10.1023/A:1010727220929

[12] ISO Central Secretary. 1993. *Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s – Part 3: Audio.* Standard ISO/IEC 11172-3:1993. International Organization for Standardization, Geneva, CH. https://www.iso.org/standard/22412.html

[13] ISO Central Secretary. 2006. *Information technology – Generic coding of moving pictures and associated audio information – Part 7: Advanced Audio Coding (AAC).* Standard ISO/IEC 13818-7:2006. International Organization for Standardization, Geneva, CH. https://www.iso.org/standard/43345.html

[14] S.M.S. Jalaleddine, C.G. Hutchens, R.D. Strattan, and W.A. Coberly. 1990. ECG data compression techniques-a unified approach. *IEEE Transactions on Biomedical Engineering* 37, 4 (1990), 329–343. https://doi.org/10.1109/10.52340

[15] Mandeep Kaur, Shikha Kakar, and Danvir Mandal. 2011. Electromagnetic interference. In *2011 3rd International Conference on Electronics Computer Technology*, Vol. 4. 1–5. https://doi.org/10.1109/ICECTECH.2011.5941844

[16] Google LLC. 2022. Protocol Buffers Documentation. https://protobuf.dev/programming-guides/encoding/

[17] Gonçalo S. Martins, David Portugal, and Rui P. Rocha. 2014. A comparison of general-purpose FOSS compression techniques for efficient communication in cooperative multi-robot tasks. In *2014 11th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, Vol. 02. 136–147. https://doi.org/10.5220/0005058601360147

[18] Inc. Nabu Casa. 2024. *ESPHome Devices.* https://devices.esphome.io/

[19] Luna Peshkov. 2024. aztec. https://github.com/Orange-Murker/aztec

[20] Luna Peshkov. 2024. Fix memory leaks. https://github.com/dblalock/sprintz/pull/5

[21] Dhawan Singh, Amanpreet Sandhu, Aditi Thakur, and Nikhil Priyank. 2020. An overview of IoT hardware development platforms. *Int. J. Emerg. Technol* 11, 5 (2020), 155–163.

[22] Eben Upton. 2023. Introducing: Raspberry Pi 5! https://www.raspberrypi.com/news/introducing-raspberry-pi-5/

[23] Martijn van Beurden and Andrew Weaver. 2024. *Free Lossless Audio Codec.* Internet-Draft draft-ietf-cellar-flac-14. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-ietf-cellar-flac/14/ Work in Progress.

[24] D. Witters, S. Portnoy, J. Casamento, P. Ruggera, and H. Bassen. 2001. Medical device EMI: FDA analysis of incident reports, and recent concerns for security systems and wireless medical telemetry. In *2001 IEEE EMC International Symposium. Symposium Record. International Symposium on Electromagnetic Compatibility (Cat. No.01CH37161)*, Vol. 2. 1289–1291 vol.2. https://doi.org/10.1109/ISEMC.2001.950633

A IMPLEMENTATION OF THE X86 INTRINSICS

A.1 _lzcnt_u32

```
uint32_t _lzcnt_u32(uint32_t val) {
    int temp = 31;
    uint32_t res = 0;

    while (temp >= 0 && ((val >> temp) & 1) == 0) {
        temp = temp - 1;
        res = res + 1;
    }
    return res;
}
```

A.2 _pext_u64

```
uint64_t _pext_u64(uint64_t val, uint64_t mask) {
    uint64_t res = 0;

    int m = 0;
    int k = 0;

    while (m < 64) {
        if (((mask >> m) & 1) == 1) {
            res = (res & (~(1 << k)))
                | (((val >> m) & 1) << k);
            k = k + 1;
        }
        m = m + 1;
    }
    return res;
}
```