ISA Transactions 92 (2019) 273–285

Contents lists available at ScienceDirect

ISA Transactions

journal homepage: www.elsevier.com/locate/isatrans

Practice article

Edge analytics for anomaly detection in water networks by an Arduino101-LoRa based WSN

Mehrdad Babazadeh

Faculty of Engineering, University of Zanjan, University Blvd., Zanjan, Iran

HIGHLIGHTS

- Sensor-side implementation of edge-based anomaly detection on Arduino101-LoRa.
- Customized Lempel–Ziv to achieve minimal resource usage and power consumption.
- Design algorithms and technics approved via various field and laboratory tests.

ARTICLE INFO

Article history:
Received 16 January 2018
Received in revised form 8 August 2018
Accepted 14 January 2019
Available online 18 February 2019

Keywords:
Anomaly detection
LoRa
Arduino101
Compression
Smart water networks

ABSTRACT

This paper presents a novel distributed data analytic architecture, and corresponding algorithms that apply to infrastructure anomaly detection. The proposed method mainly focuses on smart water networks, demonstrating that the highest possible sensor rate analytic performs at on-edge nodes without requiring the whole date to send back to the server. This approach saves communication costs and lengthens the lifetime of the battery-powered nodes. A complex set of tasks is developed on a single-core Intel Curie processor, Arduino101 and the raw sensor data is compressed using a customized Lempel–Ziv compression algorithm tailored to resource-constrained embedded systems. The compression rate figures are then analyzed but only the compressed data which is associated with the anomalous condition is sent back to the server by means of a LoRa platform. The developed system is evaluated experimentally and the results verify the high resource utilization.

1. Introduction

The development of wireless sensor network (WSN) platforms to support critical, potentially inherently vulnerable and costly infrastructures, such as water networks is gaining much interest due to their ability to monitor utilities to provide early warning of deterioration or failure (e.g. leakage). In traditional monitoring systems, such utilities may use expensive wired platforms [1] which are usually integrated into a supervisory control and data acquisition (SCADA) system. However, distributed wireless monitoring systems are considered more desirable due to their flexibility and cost-effectiveness and ability to be retrofitted in difficult to reach or dangerous areas. Prior research, e.g. [2] refers to water network infrastructures and modeling and [3] that focuses theoretically on fault detection in water networks using modeling and state estimation methods. As technology advances the complexity of the onboard processing is increasing over time, for example, [4] introduces a WSN to detect and identify major anomalies in steam flood pipeline networks in two stages, single node processing, and multi-node collaboration. They build a decision tree to capture the salient pressure and flow characteristics of each problem to distinguish them from false alarms. They explain it only in a systematic way without pointing out the results and design details. A seminal article in the field of water pipeline's anomaly detection using sensor networks is [5] which describes the application of Intel Mote sensor nodes (SNs) to collect required samples for anomaly detection, transmit the Min/Max/Average data and go back to sleep. Data is relayed via the GPRS modem to a back-end server. This approach is what many subsequent researchers have followed since. However, the major disadvantage is that all sensed or aggregated data is expected to be sent to the back-end system for analysis. This means that the battery-powered nodes will deplete their resources as they have to communicate all the data to the server where it is not always necessary. Authors in [6] primarily address different pipeline leakage detection methods, the detection from inside [7] or from outside [8] of the pipes. A comparative study in [7] shows that the interior detection methods are more accurate than the outer detection methods. Therefore, [6] provides a wireless communication system for underground pipeline inspection where the sensor nodes inside the pipeline are mobile and carried by robots. The system includes portable sensor nodes inside the underground pipeline, aboveground relay nodes deployed along the pipeline, a remote monitoring center, and a mobile communication network from a third-party provider. They also mention

E-mail address: mebab@znu.ac.ir.

https://doi.org/10.1016/j.isatra.2019.01.015
0019-0578/

Nomenclature

ISM	Industrial, scientific and medical radio
CR, *FCR*	Compression rate, Filtered compression rate
LZ	Lempel–Ziv data compression method
LPWAN	Low power wide area network
RTC	Real time clock
T_s	Sampling time
T_{wait}	Waiting time in the wait loop
T_{R1}	Total reading and transmitting time of a compressed packet
T_{R2}	Pure data transmission time
S	Laplace operator
LoRa	Long range communication technology for IOT
Z	Discrete time shift operator
T_{comp}	Time needed for data compression
T_{rest}	Execution time of the rest of the main loop
$T_{task(n)}$	Execution time of the task number n
S_W	Overall writing speed to SD memory card
$t_{SDfill(min)}$	Total minimum time to filling up SD card with data
$U(S)$	Laplace form of the raw measured signal
τ	Time constant of the first order low pass filter
u_t	The raw measured signal in the time domain
$Y(S)$	Laplace form of the filtered signal
y_t	The filtered signal in the time domain
Z	Discrete time shift operator
P	Average power (W)
V	Average voltage (V)
I	Average current (A)

that the sensor nodes are equipped with different types of sensors, including acoustic or pressure sensors to detect a leak. However, the way to detect a leak is not discussed. This potentially expensive and complicated method might work under special circumstances where there is a control on the robots carrying the nodes and bidirectional communication between the sensor nodes inside the pipe with outside, but it may not be suitable for all pipe size and type. Reference [9] presents a linear WSN application on the liquid protection systems where the main focus is on the energy aspect. They introduce an energy-efficient node placement scheme where the optimum node number and nodes' distance is calculated but with no discussion on detecting anomalies in the pipes. Another article [10] simply uses the CC2530-based sensor node for real-time monitoring and data collection such that the warning is transmitted in case of anomaly occurrence and make the alarm decision according to the pre-stored pipeline safety database. This method may work in very simple scenarios but not in the real water networks since it does not have any fixed structure and ability to determine the safety database. Moreover, the limited communication range of CC2530 might be another practical flaw.

Nowadays it is possible to use much finer grained and higher fidelity sensors to provide more information about what is happening to the pipes (e.g. the NEC Tokin vibration sensor can reach 10 kHz). This yield to the ability to detect issues much faster and earlier. However, the cost in terms of battery depletion and communication over a data network to send such data is prohibitive. Therefore, practical data management and edge processing by embedded systems are crucial to both minimize communications and in doing so reduce the complexity of back-end processing meaning that analysis can be carried faster and in real-time. The reduction of the data in a methodological and efficient way is the focus of this article aiming to minimize the payload for communication in a wide-area water network.

The main contribution of this paper is to propose and implement new optimal algorithms for the anomaly detection on single-core processors, integrate LoRa technology, and finally test and verify the results. This is the first work to demonstrate the complex multitasking operations on simple low-end MCU based sensor nodes to process high-fidelity sensor data at the edge. The primary stage of the work with limited results was done by the author and presented in [11] for the European designated ISM band of 868 MHz. The idea of using the data compression rate (CR) for anomaly detection has already been proposed in [12]. It is realized throughout the present research work considering many technical constraints toward making a wide range and low power edge-based anomaly detection system. This article reports important improvements by optimizing the implementation and maximizing the payload size. The resultant attractive outcomes applicable for all ISM bands and the Asian ISM band of 433 MHz in particular is also presented.

The paper is organized as follows: Section 2 briefly refers to the system overview, requirements and practical limitations. The next Section 3 refers to the methodology implemented for the anomaly detection. This section describes the pseudo codes for the proposed algorithms with several practical considerations and discusses different tasks in detail and represents an important timing diagram that clarifies the whole system operation. The last Section 4, states the performance analysis for the most important tasks associated with the possible scenarios. The last subsection represents the power consumed by each SN during the defined worst-case scenario.

2. System overview, requirements and limitations

Fig. 1 illustrates a symbolic architecture which covers a section of a water network equipped with a WSN with tens of SNs located in a 3–5 *km* circle area, a base station or central processor where further data evaluations are carried out, and the cloud to keep data available through the Internet. In the network highlighted by the red color, a group of the local SNs (SN_1, SN_3, and SN_4) sense the water pressure variations, that might be because of occurring anomalies (water leakage or burst) in the pipes or other equipment. In order to detect such failures or malfunctioning, the defined system has to fulfill the following requirements:

- Measured signal is sampled at the maximal achievable sampling rate for the sensor and the node system that can be handled based on the developed hardware.
- Data is temporarily stored depends on the node's available amount of storage, then compressed and finally, edge-anomaly detection is carried out based on the CR [12].
- The utilized data compression technique should be lossless to maintain data quality.
- upon detection of an anomaly, the originating nodes send immediate notifications to the central, indicating the source node identity and time detected.
- The server subsequently asks for extra data from the source nodes. It then sends this as a request to the source nodes with a schedule, indicating when they should transmit the data back to the central base station. Nodes not seeing the event remain silent.

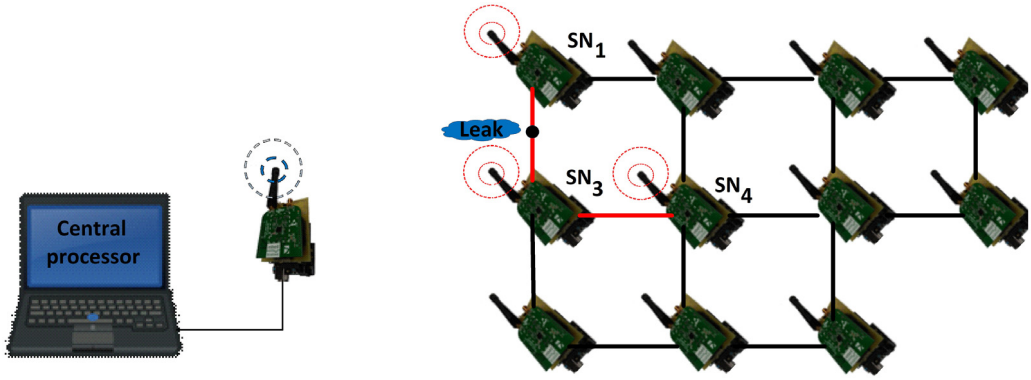

Fig. 1. Water network architecture with sensor nodes, central node and server.

- The source nodes will communicate a string of compressed data packets, including before/after anomaly and also the corresponding timestamps.
- The compressed data on the server side is decompressed to generate original pre-filtered data for further evaluation of the anomalies.
- The event connection to the central system uses a carrier sense MAC protocol with no delay in sending data. It is used because the nodes are sparse and the interference should be low, so this approach minimizes delays indicating an anomaly.
- The permissible ISM band for this research is 433 MHz.

3. Implementation of anomaly detection

3.1. Proposed Arduino101-LoRa based WSN

Arduino101, a low-power consumption Intel-Curie module has been used as the base platform for the implementation of the SNs in this research due to its unique characteristics. The 32-bit processor with 24 kB SRAM and 196 kB flash memory [13] facilitates the design setup with commercially available peripherals, but hard considering its single-core processor's limitations. It is an example of a low-end, MCU-based node but the work is not limited to this architecture and the other MCU-based nodes can also be used. The proposed distributed WSN looks similar to a parallel computing at the system level. However, all tasks have to be implemented sequentially in the SN level on this processor. In summary, being the single-core and having SRAM limitation are the main constraints for the implementation which are discussed further in the next sections. Fig. 2 shows the implemented SN(right) and input–output connection diagram (left) which makes the appropriate connection to analog sensor/s and other peripherals (middle) as follows:

- LoRa shield (SX1276MB1MAS): Used to achieve a wide-area communication [14].
- SD card shield ('Stackable SD/TF Card Shield V3 for Arduino'): For the read/write cycle on the SD memory card.
- Real-time clock (RTC) module: To have an accurate clock time for SNs.
- Extension board: Designed and developed to connect analog sensors and RTC module to the Arduino101 and to make the appropriate connection to the LoRa module and SD memory card shield.
- Pressure sensor: As the candidate of a fast analog sensor to get water pressure measurements, a *PT*4400 can be considered with the detailed specifications written in [15]. To be independent of the water network while developing the platform, a light sensor is primarily used instead of the pressure sensor where it provides similar measured signals for the rest of the process.

3.2. Anomaly detection procedure

Anomaly detection program includes different tasks located both in the main thread, and a callback function in the second thread. The below Algorithm 1 represents the pseudo code for the main thread in the Arduino-based anomaly detection system. The wait loop in the first thread ensures the input data array is filled up with pre-filtered sensor data. The array pipelines that data to the next important stage, the data compression algorithm.

The number of interrupts during the loop execution depends on the maximum time required for each task in the loop and the defined sensor sampling time (T_s). Due to have several fast and slow tasks and to deal with the concurrence, an interrupt service routine has been used as an additional pseudo-thread represented in the Algorithm 2. The interrupts may occur everywhere during either the wait loop or the infinite while loop operation. Since the SN-side tasks in the single-core processor have to be executed consecutively with the interrupt/s in between, it is important to consider both individual and the total required time of tasks.

Data: Set up input data, arrays and Interrupt Timer
while do
  while *input for Comp.algorithm is not complete* **do**
    Wait and call Algorithm 2 every T_s millisecond;
  end
  Call Compression Algorithm; Determine CR; Apply a low pass filter to CR;
  if *received any synchronization Request* **then**
    Update RTC;
  else
    Read RTC;
  end
  timestamp = RTC time;
  if *there is Anomaly or it is inside timer1 period* **then**
    Start timer1 by the first anomaly; Count timestamps until timer1 = 1 min; Go to Algorithm 1 look for the worst anomaly for 1 min; Notify Center at the end of 1 min period; Reset timer1 and Start timer2;
  end
  Append timestamp and data packet to SD memory card (refer to Algorithm 3);
  if *There is a Data Request and* T_{timer2} $<=$ 1 *min* **then**
    Read SD card and transmit data (Algorithm 4);
    Reset timer2;
  end
end

Algorithm 1: Pseudo Code for the main loop of SN

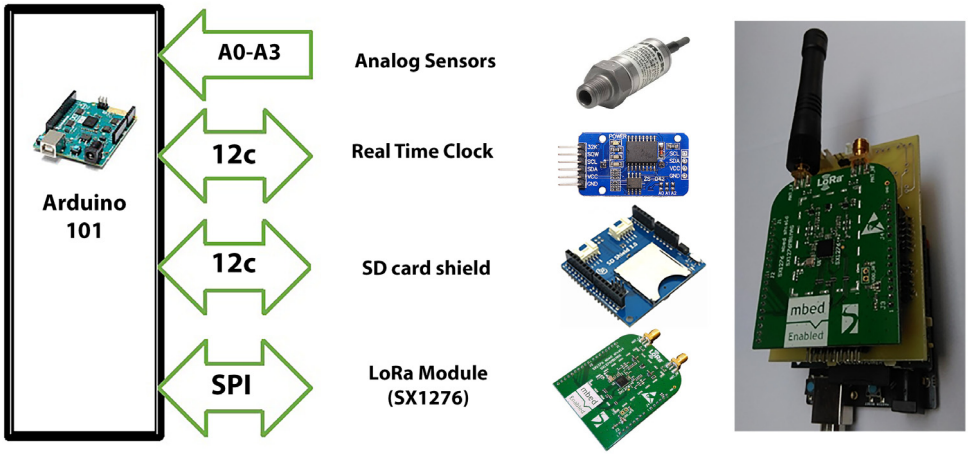

Fig. 2. Developed Arduino101-LoRa board and I/O connection diagram.

Read Sensor;
Apply sensor Transfer function;
Apply pre-filter and put new data in the input data array;
Back to where interrupt started in Algorithm 1;

Algorithm 2: Pseudo Code for Callback function

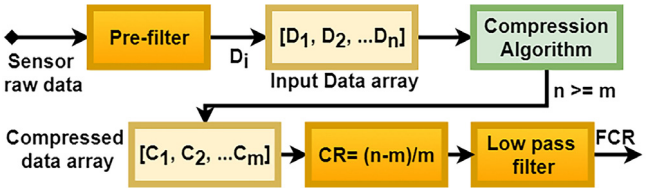

Fig. 3. Data flow from the measurement point to filtered compression rate.

Properly scheduling tasks to the above-mentioned threads and considering their execution times helps to cover all the requirements and optimize the whole system. Otherwise, the system may fail due to the low SRAM capacity, inappropriate memory and/or time allocation. The anomaly detection system includes several major tasks, which are discussed in the next subsections.

3.3. Major tasks in anomaly detection

3.3.1. Lossless compression algorithm

Applying a compression algorithm to an input data array, the data compression rate (CR) is calculated according to the length of both input array and the associated output array as represented in (1):

$$CR\% = 100 \times [\frac{input\ length - output\ length}{input\ length}] \quad (1)$$

Because of the limited memory size of the processor assigned for an input data array, the input length supposed to be fixed. However, the output length will vary according to the nature of the data in its corresponding input data array and is not fixed.

According to [12], the anomaly detection carries out upon a correlation between raw data value fluctuation and the corresponding CR. This scheme enables identification of transients or failures in the systems such as water distribution networks where they can be detected by evaluation of the CR instead of the raw measurement data. The local detectors, SNs in this scheme neither transmit raw data nor compressed data to center for primary anomaly detection. They evaluate the compression rate to detect anomalies and outliers first and notify the center afterward. The center will ask for the anomalous compressed data for further evaluations in time. This will offer a low power consumption WSN resulting in an early detection of anomalies with faster and more lightweight procedure compared to the usual methods.

To find the best compression method, several small-sized algorithms were tested during this research. They did not succeed due to the limited SRAM space available. The Lempel–Ziv algorithm, developed in [16] and addressed in [17] was also tried out since authors in [18] had reported they had implemented a lightweight subset so-called Mini-LZO on their Intel Edison platform with 2 GB SRAM. However, it is required memory size exceeded Arduino101's 24 kB SRAM. Another function, LZO1X_1_11 was found in the original LZO library and customized by manipulating the memory allocations and the library modules to be tailored to the memory-constrained Arduino101.

Increasing the input data array size in the proposed system with the sequential tasks can improve the compression performance provided that a sufficiently large sampling time is set to reserve enough time to execute other tasks in the main loop. However, the slower sampling rate means the lower system performance which is not acceptable. Optimizing the SRAM usage was carried out along with integrating the other tasks and by compromising the measurement speed, SRAM space, and compression performance and considering future developments, the input size was set to be 1 kB. A high compression rate (CR% close to 100 %) obtains for input data windows with the low data variation and lower CR% reflects more input data variations.

3.3.2. Sensor data reading and pre-filter

Fig. 3 illustrates the data flow diagram from the sensor measurement point to the filtered compression rate. The sensor signal is sampled every T_s millisecond by executing the second thread which is a callback function represented by the Algorithm 2. To have one-byte data format, the sensor data scales between 0 to 255 as unsigned character. This minimizes the memory usage and maximizes the number of input data for the compression algorithm. The sensor measured signal varies because of the sensor noise and/or different interactions within the water network such as the system response in the water demand and resource changes, anomalies, etc.

The length of the output compressed data array depends on the measured data variations in the input data array. Since the small variations in the signals lower the compression performance, using a pre-filter can improve the performance, prepare the noiseless

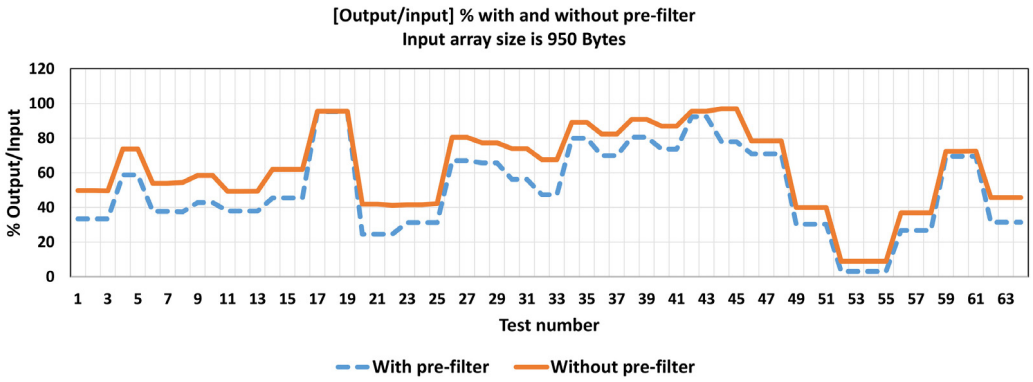

Fig. 4. Rate of reducing data size with and without pre-filter after applying compression algorithm.

data for the process, and reduce the total payload for the compressed data communications. As the low pass pre-filter, a moving average or a first-order filter do not change the fundamental form of the measured raw signal. The Laplace form of an implemented simple first-order low-pass filter and its discrete time equivalent driven based on the Backward Euler method is given in (2) and (3) where S, $U(S)$ and $Y(S)$ are the Laplace operator, raw and filtered signals respectively and Z is the discrete time shift operator. τ, u_t and y_t are the time constant, input and output in the time domain respectively.

The sampling time in the filter sets according to the required bandwidth of the measured signal and other processing limitations that will be discussed later.

$$\frac{Y(S)}{U(S)} = \frac{1}{\tau \times S + 1};\ S = \frac{1 - Z^{-1}}{T_s} \tag{2}$$

$$y_t = \frac{1}{\frac{\tau}{T_s} + 1} \times \left(\frac{\tau}{T_s} \times y_{t-1} + u_t\right) \tag{3}$$

Fig. 4 represents the percentage of the compressed output data size out of the input for a real test case with and without the pre-filter. The gray solid graph points out the rate without applying the pre-filter, and the dashed line illustrates the same factor after a low pass pre-filter applies. The average compacted data size without the pre-filter reduces up to 51% in comparison to the size of the input data array at the same time, applying the pre-filter to the identical test case results in a 23% more reduction in the compressed data sizes. This yields better internal and external memory usage, the lower communication payload, the shorter time of the slow LoRa communication, and the lesser total power consumption of the WSN which is the main objective of this research.

3.3.3. Using filtered compression rate for anomaly detection

The abrupt extreme anomalies can dramatically change the raw measurement values and therefore the CR profile. By looking at this profile, the detection system can figure out the occurrence of either suspicious events (transients) or anomalies. The CR associated with the measured data of normal operations should have less variation than abnormal situations. Applying an additional low pass filter to CR can smooth the CR profile. Assuming that a pre-defined threshold is a border to distinguish anomalies from normal operations and unwanted transients, this low pass filter can remove transients and keep CR associated to the normal operation even further from the threshold. Therefore, the probability of catching the correct anomalies increases. A solid blue graph in Fig. 5 illustrates an example of a normal CR with great deal of variations. The corresponding filtered data compression rate (FCR) by applying a first order low pass filter is also represented by a smoother dash gray color. In a simple definition of anomaly detection, an anomaly can be detected when the FCR falls below a well-defined definite or adaptive threshold [18]. The red dash–dot line in this figure denotes a predefined threshold, the borderline for the detection process. It is noteworthy that the indicator time (t = 70 s) represented by the FCR might slightly differ from the accurate time of the anomaly within the raw data. The initial value for the readout (FCR) sets to 100 to stay above the threshold to avoid issuing the wrong anomaly when the system starts up. The value of the time constant (τ) in this filter depends on the value of the signal variations and it has been set to 0.2 in this example.

This is the primary stage of the anomaly detection in the SN-side and further evaluation can be carried out in the center after the pre-fault and post-fault data are decompressed which can be the subject for another research.

3.3.4. Writing to SD memory card

It is useful in many applications to keep the long-term data in real time when they measure at the highest possible frequency. The SD memory cards are slower than the internal non-volatile memories and the low speed of the writing process to the SD memory cards limits their fast real-time usage. However, it can be used appropriately to keep compressed data with a shorter size in comparison to the long raw data arrays. In this project, a Stackable SD/TF Card Shield V3 for Arduino is used together with a fast SD memory card. As [19] lists, there are many alternatives to choose one among different viable SD memory cards and a 16 GB Class 10 Ultra SDHC UHS-1 Memory Card — 80 MB/s has been used primarily to achieve the fastest possible Read/Write (R/W) operations. The Arduino101 has an SD memory card's R/W library (SD.h) which is already designed to append data from the first address 0 to the end of the SD memory card. However, extra control on the R/W functionality is needed to manage the variable-size compressed data packets in the middle of a text file on the SD memory card.

The most important considerations while implementing 'Writing data' to the SD card are pointed out in beneath:

- The compressed data packets have different sizes depending on the nature of the fed input data. Data is appended packet by packet, each as a string of an unsigned integer number into a text file and each number in a separate line.
- An SD card with 16 GB memory stores long time data to:
  - Access to a long history of compressed data even more than a year.
  - Read a part of data from the SD card while fast coming new data is being written.
  - Make a data string and transmit it via the slow communication platform (LoRa) while there is no risk of the data being overwritten on the SD card

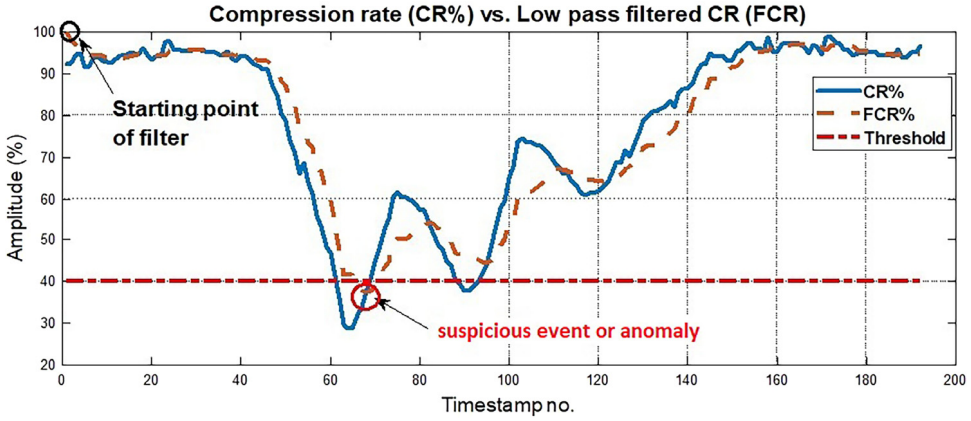

Fig. 5. Data compression rate before and after using a low pass filter.

- Memory usage of the SD card is checked while writing, and the data are entirely removed when it exceeds a predefined memory limit. Then the next data packet is written from address 0 in the SD memory card. The limit value sets to at least 1 kB less than 16 GB. Because the lengths of the packets are variable, and it should be guaranteed, the last packet is fully written at the end of the latest written data in the text file.

According to the Algorithm 3, writing to the SD memory card in every loop execution starts with opening a text file, checking the memory status, appending a combination of a timestamp, corresponding compressed data arrays, and identification numbers as shown in Fig. 6. It terminates by closing the text file. The data in the SD memory card includes a symbol, '*T*' connected to a timestamp and follows with a variable size array of compressed data. This full array (timestamp + compressed data) holds a big size together with some extra characters for separation of numbers and/or separation of lines while being written on the SD memory card.

By removing the additional spaces from the characters, they will look like connected characters and cannot be distinguished by the center after transmission. This was a problem in [11], where transmitting packets more than 32 bytes was impossible. This is addressed in the next section where the maximum possible payload, 250 bytes can be reached. After transmitting the compressed data, it is decompressed in the center. Not all the packets begin with a timestamp. For example, if a packet is longer than the predefined maximum size (250 bytes), it will start with a packet, including a timestamp and continues with the next packets without the timestamp. Therefore, there should be other flags at the end of each packet to make the center aware of the packet type. The center carries out different operations on receiving the various packet types.

Packet x				
Timestamp	Data	Sub-packet No.	Node Address	Sub-packet ID (1 byte)
4 bytes	<246 bytes Compressed data	1 byte	1 byte	1 byte

Fig. 6. An example of a compressed data packet format in the SD memory card.

Data: Address of the last written data
Result: To append new compressed data packet to a data text file
if *the text file exists* **then**
  open it;
else
  Create it;
end
if *not reached to the far permissible capacity limit* **then**
  1. Append the timestamp ;
  2. Append the data packet line by line;
else
  3. Clear all data from the SD memory card;
  4. Set writing address equal to zero;
end
Close file;

Algorithm 3: Pseudo code for writing to SD memory card

3.3.5. Reading from SD memory card

In the reading stage, a long data string is made from the relevant packets, and then the SN communicates the string to the center which is another Arduino101-LoRa based SN. Algorithm 4 represents the pseudo code for reading data line by line from the SD memory card and data transmission if requested. As already mentioned at the end of the previous section, to maximize the number of the data size of the possible payload, the data format previously written on the SD memory card has to be manipulated. It is written as characters and holds the larger size than pure data. Therefore, it is needed to separate the timestamp symbol ('*T*') and make integer data before the transmission. By this, the extra separating characters are neglected and at the same time; a 1-byte unsigned integer for each data is assigned. The communication packet size is already limited by default, and the entire communication payload size affects the total communication time and the node power consumption. Therefore, it is prominent to consider making every packet effectively. The data format must have two important characteristics. First, it must be able to be separated by the central node while the decompression algorithm is applied. Second, the timestamps must be distinguished from the other data and then extracted from the received packets before the decompression algorithm is applied.

The most important considerations while developing code for the reading and communication tasks are:

- Reading a specific number of the last compressed data packets with the full length should be guaranteed.
- It is read in the same format as it has already been written to the SD memory card. Therefore, data must not be either overwritten or read from the middle of an array. It must start with the first item (timestamp) and terminates at the end of a compressed data packet.
- The primary definition was to prepare 2 min of compressed data for communication by receiving a request from the center. However, it can be an unlimited time in the recent development proposed in this paper. After decompression at

the center, the data packets will be expanded to generate the longer filtered raw data arrays.
- If the pointer of the last data is in the early part of the text file and there are not still many data arrays to send, it will send whatever data arrays already existed from address 0 up to the present address of the writing pointer. Otherwise, it will send enough data on the stored history.

3.3.6. Data communication

There might be either short or long communication packets in the proposed WSN as follows:

- System settings packet: The different types of setting packets within the implemented WSN are initial parameter settings, anomaly data request, and synchronization command. They are short packets of a few bytes transmitted by the center to the SNs. For example, a data request packet includes the SN number and the time of the anomalous data transmission.
- Worst-anomaly packet: After the first anomaly is detected by an SN, it activates a wait timer (1 min) and the next possible anomalies during this period are evaluated. In the end, a notification representing the worst anomaly with the lowest FCR together with the related timestamp is transmitted to the center, and the wait timer is reset to be prepared for the next round of anomaly detection.

Data: Required history (time) of the stored data
Result: Make a fixed size sub-array of compressed data packet, add identification and timestamp if needed and then transmit to center
Open text file in the SD memory card;
Symbol ('T') Counter = 0;
while *pointer is less than Post-fault address* **do**
  while *Pointer has not reached to the beginning of the required packet* **do**
    Still go one-step back in the memory address toward the starting point;
  end
  if *(Symbol ('T') Counter = 0 meaning that it is the first Character equal to 'T')* **then**
    Symbol ('T') Counter++;
    Extract 'T' from timestamp;
    Separate different parts of timestamp;
    Put them consequently in the primary rows of a transmission array in unsigned integer format;
  end
  while *Pointer has not still exceeded the maximum possible payload size and not reached to the next timestamp (started with the symbol 'T')* **do**
    Add new unsigned integer data to the packet
  end
  Make an integer data string including pure timestamp and compressed data array;
  Add specific Flag to the end of the packet to show either it is last packet or not;
  if *(If it is the rest of the previous packet)* **then**
    Transmit the rest of previous packet which is without timestamp;
  else
    Transmit a new packet starting with timestamp;
  end
end
Close file;

Algorithm 4: Reading SD memory card and transmitting compressed data to center

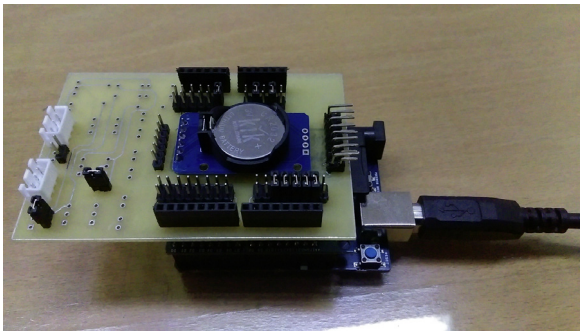

Fig. 7. Developed extension board for RTC and I/O.

- Anomalous data packet: After notifying the worst-anomaly, another 'wait until the request' timer (1 min) is activated by the source node. After receiving a data request from the center, the source node transmits approximately 2 min of the pre-fault (1 min) and the post-fault (1 min) compressed data. The data is transmitted according to the determined schedule by the center. The center might request the data from different SNs one by one (in the node-to-node implementation) or simultaneously (in the WAP implementation). For the limited number of SNs with the correct settings, the node-to-node structure should work fine enough if an appropriate time schedule is set by the center. Otherwise, a gateway is needed to manage simultaneous receiving data packets from the SNs. The data transmission phase includes the following tasks:
  - Opening the SD memory card to read the data according to Algorithm 4.
  - Looking back to about 1 min before the worst notified anomaly for the starting point of pre-fault data.
  - Looking forward to about 1 min after the worst notified anomaly for the endpoint of the post-fault data.
  - To make fixed-size sub-arrays from the compressed data between the starting point up to the endpoint.
  - To apply specific identification numbers to each sub-packet.
  - Transmitting the prepared packets one by one.

3.3.7. SN-based clock time and timestamp

Retrieving the real clock time for each SN is needed to make the timestamp and apply appropriate control in the process. Therefore, an I^2c based RTC module (DS3231) was integrated as shown in Fig. 7 to achieve an extreme accuracy. The timestamp is the initial part of every compressed data array that SNs transmit in one or more smaller packets if requested by the center. The RTC module is programmed to provide required unique timestamp, which is a combination of 'Year', 'Month', 'Day', 'Hour', 'Minute' and 'Second' data for the SN. For instance, 180228231219 represents the timestamp of the compressed data packet, set to the 28th of February 2018 at 23:12:19. This is an accurate local time for the SN while writing the last compacted data on the SD card

3.3.8. Synchronization

The SNs operate independent with their own 4 s loop time. They have 0 to 4 s clock time difference. To fix this issue, their clocks have to be synchronized by an external clock time. Since their execution is not simultaneous, they read the message broadcasted by the center at different times in the middle of their own loop. Therefore, the real clock time should be broadcasted frequently. There might be three methods to deal with synchronization as follows:

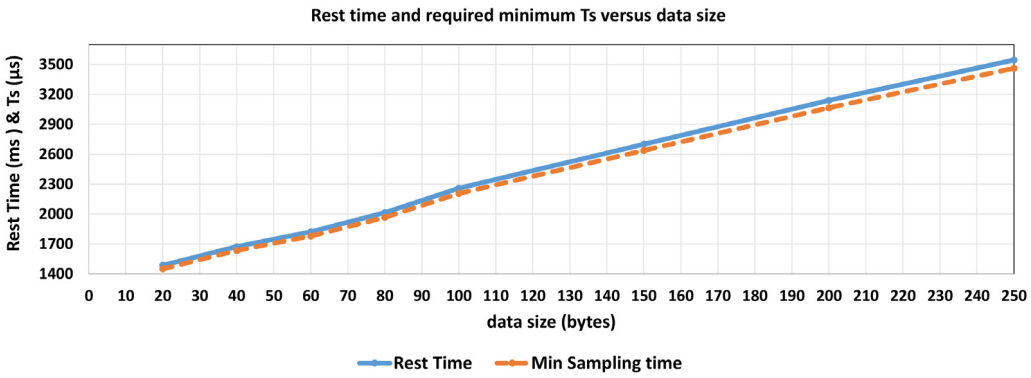

Fig. 8. T_{rest} *and minimum required* T_s *versus data size* (byte).

- Frequent synchronization packets should be transmitted by the center for at least 4 s to make sure all SNs get the right clock time and set themselves accordingly.
- To achieve more accurate clock signals, all the SN's activities are bypassed as soon as an SN receives the command signal for a while until all the SNs are synchronized and assign the main loop execution only for this regulation. The broadcasted synchronization packets have to be refreshed frequently for at least 4 s to make sure all SNs get the right clock time and set themselves accordingly.
- To equip all SNs with a GPS module to set their time according to the satellite time. This is a good choice if not only the SN can supply the GPS power, but also the extra module has a reasonable price, and the satellite signal is available for the SNs.

3.4. Worst-case scenario

The implemented simulator of the base station includes only an Arduino101, an RTC module to provide timing commands for the SNs and a LoRa shield to make a node-to-node communication possible. The measuring SNs and the central SN are equipped with appropriate antenna and they are powered via batteries and computer USB respectively.

The worst-case scenario is a rare case where there is no data compression possible at all, the compressed output length will be about the same as the input size (1 kB). Then, the maximum-size packets will be written in the SD card in every loop time. This scenario will yield the maximum time consumption for all tasks and is called the worst-case scenario in this research. Furthermore, in the worst-case scenario, SNs are set to the maximum power mode for transmission to maximize their communication range.

3.5. Main loop operation

To discuss the timing in the main loop, three parameters: T_{wait}, T_{rest} and T_{comp} are defined in below:

T_{wait} = $T_{task(1)}$: Maximum running time of the wait loop = Length of the input data array $\times T_s$.

$T_{rest} = \sum_{n=2}^{6} T_{task(n)}$: Maximum total execution time for the rest of the main loop for the worst-case scenario. The maximum likely compressed output array size is considered where $T_{task(n)}$ is the execution time of the nth task and T_{comp} is the maximum executing time of the compression algorithm. To achieve the minimum possible T_s, the following conditions must be satisfied together in every loop execution:

$$T_{comp} < T_s \tag{4}$$

$$T_{rest} < T_{wait} \tag{5}$$

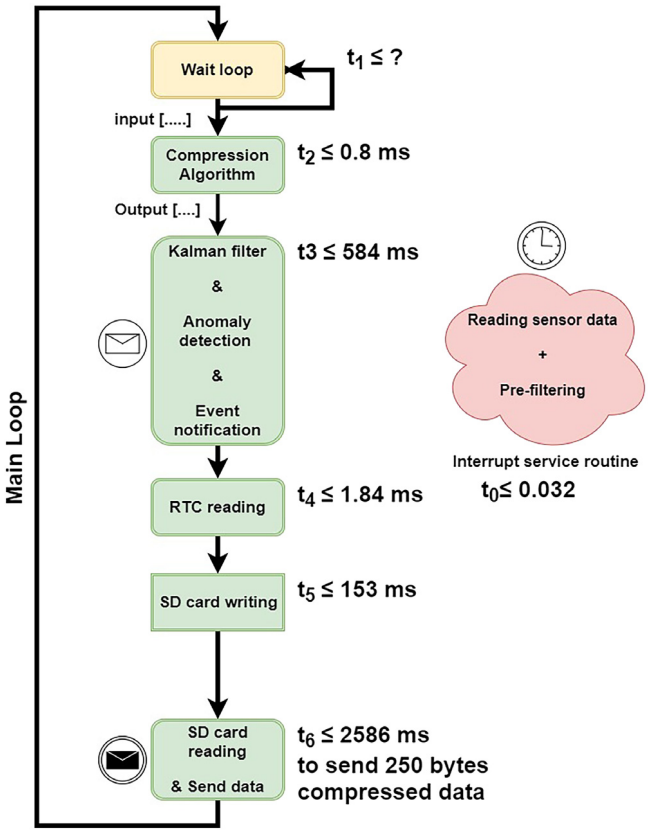

Fig. 9. Flow diagram of the maximum times consumed by different tasks.

Fulfilling the above inequalities, the main loop will run in an approximately fixed time. According to the defined worst-case scenario, for an output array with the size of 1 kB and $T_s = 2$ ms, the following values are measured:

$T_{wait} = 2$ s
$T_{rest} = 3.54$ s
$T_{comp} = 0.7$ ms

Obviously, the condition in (1) is fulfilled with the above-mentioned values. Satisfying this condition is needed because there must not be an interrupt while the compression algorithm is executing. The second condition is not satisfied while running T_{rest} = 3.54 s and there will be data overwritten before it is compressed, which is not acceptable. Fig. 8 represents the relationship between the defined rest time and minimum imaginable T_s versus data size for communication. Whatever data size is larger, both the rest time and the minimum possible T_s has to be greater as well. The

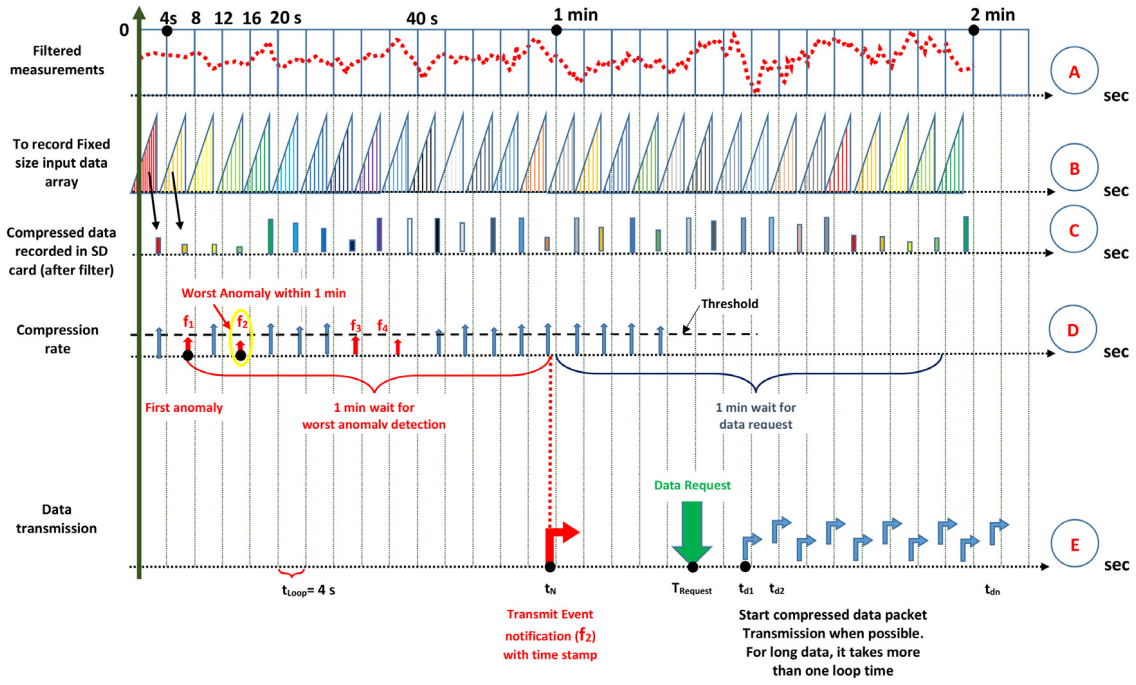

Fig. 10. Operational timing diagram of the SN-side anomaly detection.

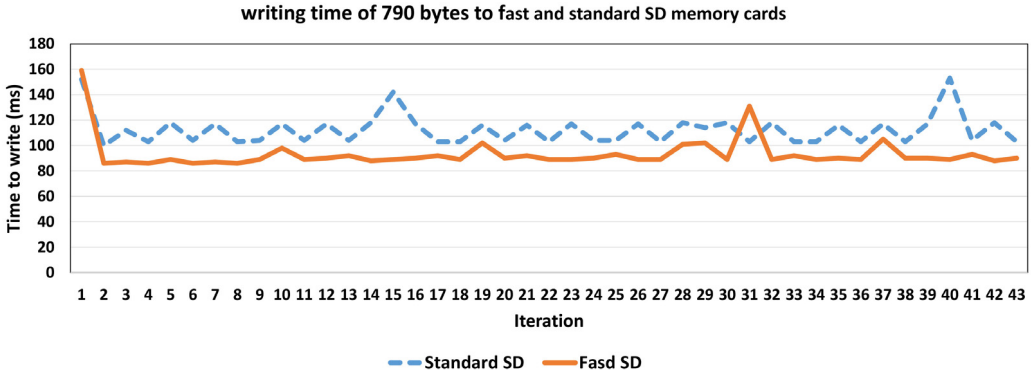

Fig. 11. Writing time of standard and fast SD cards for 790 bytes data.

maximum time required for each task in the worst-case scenario is represented in Fig. 9. In the earliest stage of this diagram, an internal wait loop that executes in T_{wait} ms ensures that the fixed size input data array (1 kB) is filled up before data compression algorithm starts. Since the main loop is executing frequently, T_{wait} depends on the number of data already filled up into the input array during the previous loop execution. It means the wait loop will run if and only if the input array has not before been filled up.

3.6. Graphical timing diagram

Fig. 10 summarizes the descriptions of the proposed anomaly detection by a graphical timing diagram. The top diagram (A) denotes a pre-filtered measured signal which is the source for the rest of the procedure. The second diagram (B) represents the increasing number of the data in the fixed-length input arrays during every loop execution time (4 s). A filled-up array is compressed and appended to the SD memory card data. Then, the input array restarts to fill up again before the next compression takes place. Diagram (C) shows the size of the compressed data arrays where the CR value can be calculated based on (3) using the fixed input array length shown in diagrams (B) and the related output array length on diagram (C). Another diagram (D) illustrates the FCR value after the low pass filter is applied to the CR. Amount of FCR, less than a predefined threshold value represents anomalies.

After the first anomaly is detected, the worst anomaly (smallest FCR) is selected among the other possible anomalies in a period of 1 min as shown also in diagram (D). Then, the SN transmits the worst anomaly information to the center as shown by diagram (E) and the center is given 1 min deadline to send its data request.

By receiving the request as shown on this diagram, the SN starts to transmit the first sub-array at t_{d1} and continues to transmit the rest of the sub-arrays during the next loops at $t_{d2}, \ldots t_{dn}$ until all the data sub-arrays are transmitted.

4. System performance

4.1. Performance of data writing

The writing to the SD memory card is the third slowest task in the present implementation. Fig. 11 compares writing speed for the fast and standard memory card while 790 bytes of unsigned char data is written to either SD cards. As seen from this figure, the fast SD memory card is approximately 20 ms faster than the standard SD card and it takes $\text{Max}(t_{task(6)}) = 152$ ms to write the long packet.

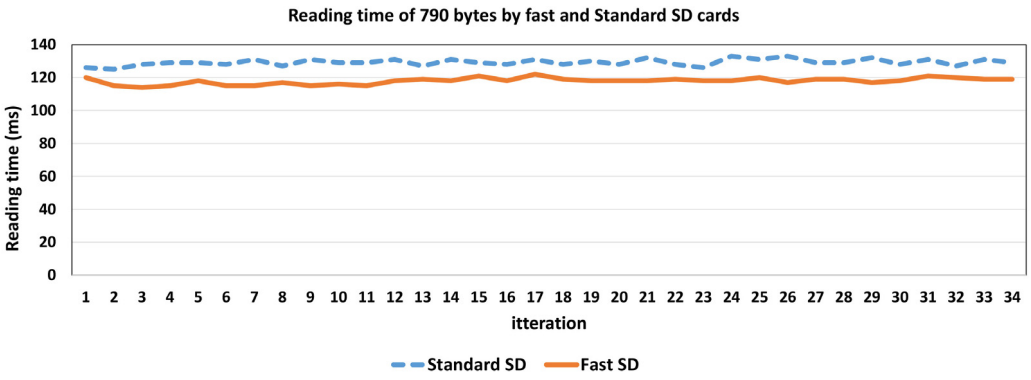

Fig. 12. Reading time of standard and fast SD cards for 790 bytes data.

In practice, for a 1 kB compressed data written on the SD memory card, the same occupied memory space cannot be expected. It is instead:

$$1\ \text{KB}\ \textit{unsigned char data} \equiv 4.37\ \text{kB}\ \textit{on SD card} \tag{6}$$

This means that due to having metadata, which is written together with each unsigned char data, the text file size is about four times bigger. Therefore, the overall writing speed (S_W) to SD memory card can be obtained as:

$$S_W = \frac{4.37}{1 \times 152 \times 10^{-3}} = 28.75\ \text{KB/s} \tag{7}$$

This is slower than the expected written speed of the fast SD memory card (80 MB/s) which is due to the overhead of opening and closing the text file, formatting, and parsing texts on the Arduino. Although another more advanced SD-Fat library may help to speed up the writing procedure, it will need more SRAM, which is not available on the Arduino101. Theoretically, every 4.37 kB data is written in each loop time in the proposed worst-case scenario. Measurements show that after 1 min, 70 kB of compressed data is written on the SD memory card. Consequently, a 16 GB SD memory card can be filled up at least in 166 days as calculated in beneath:

$$t_{SDfill(min)} > \frac{16\ \text{GB}}{70\ \text{KB/min}} \equiv 166\ \text{days} \tag{8}$$

where $t_{SDfill(min)}$ refers to the minimum time needed to fill up an empty SD memory card of size 16 GB. This duration for compressed data writing is equivalent to a longer raw data if it is decompressed. In practice, the more stable operation of the system, the better data compression and the longer t_{SDfill}. It is because the output (compressed data array) size will be smaller than 1 kB for a fixed loop time and will take longer until SD card is filled up.

4.2. Performance of data reading

Fig. 12 compares the reading speed for two types, fast and standard memory card when 790 bytes of unsigned char data, representing the worst-case scenario is read from either SD cards, and a string from the data arrays is made.

$$T_{R1} = \frac{250 \times 4.37}{2.586} = 422\ \text{bytes/s} \tag{9}$$

As seen from this figure, the average speed will lower about 10 ms by using the fast SD memory card in the proposed application. The packet size of unsigned char numbers is different from the text file size of these numbers when they are seen in the SD card text file. Therefore, two values associated with the data transmission rate can be introduced. The first value represents the transmission rate according to the size of the data file. This reveals the average speed (T_{R1}) of the whole system, including SD memory card. Every 250 bytes of unsigned char numbers occupies about 250*4.37 = 1092 bytes and needs 2.586 s to be read from SD memory card and then transmitted. Then: The second value gives the effective data transmission speed (T_{R2}) according to the pure data size as follows:

$$T_{R2} = \frac{250}{2.586} = 96\ \text{bytes/s} \tag{10}$$

The required time of the reading task varies depending on the packet size of the compressed data. After an SN receives a data request from the center, it starts to search back in the SD memory card data. After it finds the starting point, it reads one packet of data and makes a data string of integer numbers. Therefore, the maximum data size is obtained while running the worst-case scenario by manipulating the algorithm to generate the maximum output size, which means no data compression at all.

According to the proposed worst-case scenario, within the transmission procedure, a complete packet includes both the timestamps and arrays of pre-fault and post-fault compressed data, each array with the maximum possible size (1 kB) and the transmission time will be maximum. Due to the limited payload sizes available, every long array is divided into multiple sub-array to be able to transmit. In the simplest case, every full packet includes:

- A timestamp (7 bytes) only in the first sub-array.
- A pure data array in all sub-arrays.
- The transmitted sub-array number (1 *byte*) in all sub-arrays.
- An identification number (1 *byte*) in all sub-arrays indicating either the packet is the last sub-array or not.

According to the above data arrangement and considering the maximum possible communication payload size of the LoRa module, the payload size is considered in this research equal to 250 bytes. Since the remaining packets do not include the timestamp, the maximum pure data size of each packet of a long size packet can be calculated as follows:

Pure data size in the first sub-array: $250 - 9 = 241$
Pure data size in the next sub-arrays: $250 - 2 = 248$
Number of next sub-arrays $= \frac{(1024-241)}{248} = 3.15$

Therefore, besides the first sub-array, which includes the timestamp, there will be three other sub-arrays to be transmitted sequentially. The last transmitted sub-array of each packet is of size ($0.15 * 248 + 1 \approx 39$ bytes). The size of different sub-arrays of a single packet: 250 - 250 - 250 - 39 bytes and the size of pure data in each sub-array of above packet: 241 - 248 - 248 - 37 bytes. The data transmission in the present application is limited by two factors:

- The packet size limitation which is dictated by the LoRa specifications.
- The sending time limitation which applies to this sequential implementation.

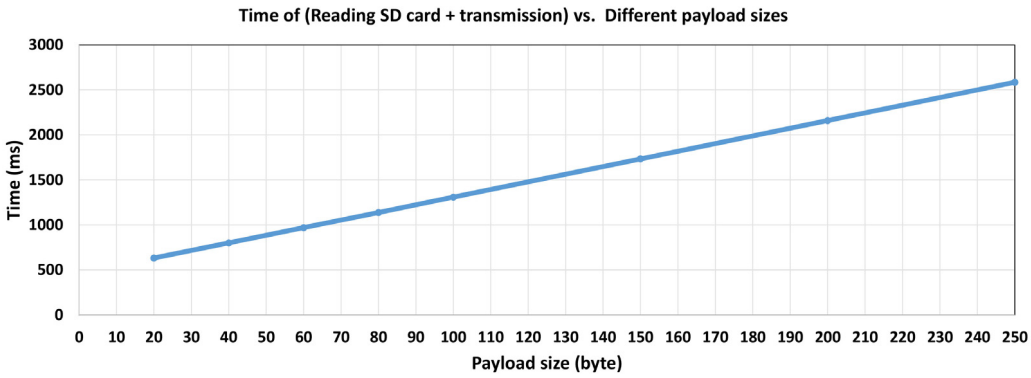

Fig. 13. Required time for both reading SD card and data transmission for different packet sizes.

Obviously, with the limited LoRa transmission rate (0.3 to 50 *Kbps*) or even lower rate respectively mentioned in [20,21], the measured raw data with $T_s = 5$ ms may not be transmitted in the real time since the following requirement is not fulfilled:

$$\textit{Transmission rate} > \frac{1}{T_s} = 200 \text{ bytes/s} \approx 1.56 \text{ Kbps} \tag{11}$$

The data packet available for anomaly detection in the present application is a 2 min combination of the variable-size compressed data arrays. The compressed data cannot be transmitted during the limited time of $Max(\Delta t_6) \approx 3$ s and will exceed either of the packet limitations defined by the LoRaWAN. This has been tested for an array of size 250 bytes as shown in Fig. 13 which takes $\Delta t_6 = 1238$ ms. This test was done using the node-to-node system where packets longer than 250 bytes could not be received by the receiver node. The compressed data packet size directly depends on the way the compression algorithm is used. By a given input length, the normal LZO compression algorithm allocates a maximum fixed output length according to the following formula:

$$\textit{Allocated Output Length} = \textit{Input Length} + \frac{\textit{Input Length}}{16} + 64 + 3 \tag{12}$$

Above relation shows that the maximum allocated output array size is greater than the maximum input size. The existing maximum available SRAM limitation must be taken into account in any kind of implementation that changes memory allocation. For example, 'Input Length = 1024' is set and from (12), the maximum total memory allocates to the Input Length and output length must not exceed 1696 bytes.

Every compression output array is appended to the SD memory card after the corresponding timestamp. If the combination exceeds the maximum payload size (250 bytes), it has to be divided to appropriate sub-arrays while transmission.

4.3. Performance of anomaly data communication

Minimization of the time and the payload of communication is of critical importance in WSNs since power, and times are limited. Besides the single anomalies, the occurrence of the severe and redundant anomalies should be taken into account to avoid transmitting the successive event notifications. Among several anomalies occurred during a specific period of time (1 min), the worst anomaly as the one with the lowest FCR is transmitted together with its associated timestamp. In addition, a source node should transmit a 2 min compressed data to the center which is impossible all at once in a limited time of the main loop execution (4 s). Therefore, a data packet with a permitted length is transmitted in each loop execution time, and the full data will take a longer time to be transmitted. Having such a long-lasting communication for the preferred anomalies, the SN may lose other upcoming detected anomalies during communication. To achieve the lossless redundant anomaly detection, the first and the last address of the packets associated with each selected anomaly can be stored for the future communications in a queue. This ensures that the base station gets all the required data of consecutive anomalies.

4.4. Performance of compression algorithm

To find the best way to control the payload size, the following options in the compression stage are discussed:

4.4.1. Normal compression algorithm (implemented)

The input data array size is fixed to 1 kB, and the output size is variable. Then, two following alternatives can be applied:

- To cut every long outgoing packet, the data string, to multiple sub-array and transmit one by one. At the end of transmission, a finishing bit informs the receiver that the full packet communication is finished. The base station in this scenario has to be smart enough to distinguish different sub-arrays of the packets to collect and re-build a correct one. Despite being more complicated implementation, it guarantees that all the packets are allowed for communication in terms of the packet size even with the lowest transmission rate. Compressing the normal data results in a short compacted data, which is not needed in the communication. However, the defective data after being compressed to build longer size arrays. Therefore, considering the maximum allowed payload size equal to 250 bytes, faulty compressed data will usually exceed this size and have to be divided to sub-arrays before communication. This causes that most of the time the full-range data of possible payload size is transmitted.
- The compressed data array sizes must be kept lower than the limit, 250 bytes. Therefore, the pre-filter and the additional low pass filter have to be applied more strongly to smooth the variations. Although this option introduces a smaller compressed data size for communication, it will result in a conservative and insensitive anomaly detection system.

4.4.2. Adaptive compression algorithm (evaluated)

According to this scenario, the size of the input data array to the data compression algorithm has to vary until it causes an output array with specific size, for example, 250 bytes. Consequently, the execution time of the compression algorithm will vary depending on the number of iterations. This is an attractive option in terms of making a full control over the communication packet size with a fixed data format.

While occurring anomaly, the data fluctuates abnormally resulting in a longer compressed data and a lower CR. In such cases,

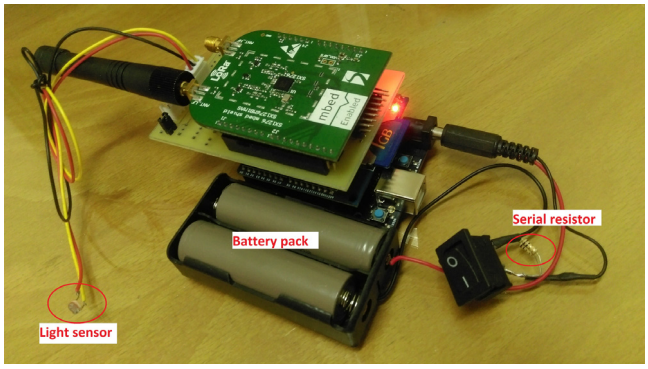

Fig. 14. Test setup to measure power consumption of the proposed SN.

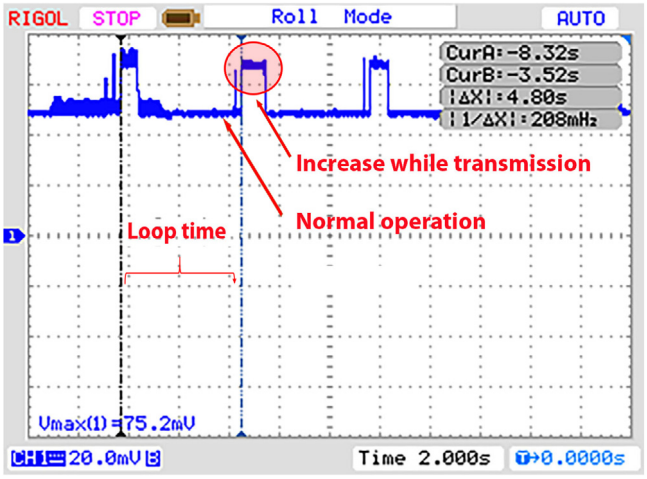

Fig. 15. Voltage drop on serial resistor for SN while working in the worst-case scenario.

the size of the input data array has to increase until it makes the same output size as it was before the anomaly. This will need more SRAM space to be allocated, which is impossible in this implementation with the Arduino's limited SRAM. Since a fixed size input and output arrays are defined at the beginning of the LZO algorithm, the utilized output size is kept as low as possible to save memory to allocate the maximum permissible size of the input data array. At the same time, it should start the compression algorithm with the least number of input data so that it does not generate larger output than the already allocated output array. The system has to increase the size gradually until the compression algorithm provides the desired output size. An upper limit (stop point) is also needed to limit the number of the utilized input data during the algorithm's execution when the input data varies dramatically resulting in a very low CR. This extra limitation must be considered as the manipulation of the compressed data and the data will no longer be the representative of the real data after it is decompressed in the base station. In addition, a tolerance for the output size has to be taken into account for the cases where the desired output array size is unachievable. Although a fixed size compressed data is generated by the adaptive approach, achieving the same CR is impossible by changing the input size, and it will vary the sensitivity of the CR as well. This is because a specific anomaly among a short and long data introduces different CRs. Therefore, if the input size is not fixed, the meaning of the CR will not be fixed anymore and will cause more complexity to the anomaly detection procedure, and this option is not implemented in the present research.

4.5. Power consumed by SNs

Reducing the SNs' power consumption is a critical and challenging task because they are operated by limited battery power. According to Fig. 14, to measure the power consumption (P) of each SN during the worst-case scenario circumstances, the input voltage (V) and the sink current (I) are used in $P = V \times I$. In this situation where the longest data packets transmit at the end of every 4 s loop time, the total power consumption of the SN is maximized and all the time consuming internal loops in the SN-side program are working to take maximum processing power. Two Lithium/Ion batteries in series connection derive 7.2 V at the total terminal voltage during the test. Fig. 15 represents the voltage drop on a 0.5 Ω serial resistor (V_R) and the current is calculated based on $I = V_R/R = 75.2\ \text{mV}/0.5\ \Omega = 150.4\ \text{mA}$ while transmission is in place. Taking more advantage of the LoRa technology, Fig. 15 represents that the amount of increasing the power consumption due to communication is less than the total processing power which is remarkable. Therefore, further attempts to reduce the total power consumption of the developed Arduino101-LoRa based

Fig. 16. Test of communication range in the worst case scenario.

SNs should target the utilized processor, peripherals, and applied algorithms instead of communication which can be a plan for the future research. This test has been done in a mountain area in Zanjan province in Iran where two Arduino101-LoRa based SNs could communicate and reach maximum 3.5km distance as seen in Fig. 16. As a complementary result of the power consumption test, the following are obtained:

- $I_{(Arduino101)} = 100$ mA ; continuous current for processing.
- $I_{(Arduino+SDcard+RTC)} = 118$ mA ; while reading/writing from/to SD card.
- $I_{(Arduino+SDcard+RTC+LoRa)} = 150$ mA ; only while transmitting.

5. Conclusions

This article revealed practical details of an optimized dual-thread implementation of a new edge-based anomaly detection on the Arduino101-LoRa basis wireless sensor network for all ISM bands and for 433 MHz in particular. As a vital part of the system, the portable lossless data compression library, LZO was customized to be implemented in the limited-resource sensor nodes. The sensor nodes evaluate their own data compression rate instead of the raw data in order to detect anomalies. The compressed data usage could minimize the needed storage space, total power consumption, and also the time and payload of the communication. The time tests guaranteed the system operation under the defined worst-case scenario and thereafter, resultant practical limitations were identified. The paper evaluated different alternatives and came up with the verified solutions.

Acknowledgments

The advanced part of the present research work has been carried out at the University of Zanjan/Iran with support from Green Power Control Co.LTD, Iran (2017–2018). The author would like to give his gratitude to the NEC Japan to fund the primary stage of the work at Imperial College London/UK (2016–2017).

Conflict of interest

The author declare that there is no conflicts of interest to this work.

References

[1] Raciti M, Cucurull J, Nadjm-Tehrani S. Anomaly detection in water management systems. Crit Infrastruct Prot 2012;98–119.
[2] George R. Critical infrastructure protection. Int J Crit Infrastruct Prot 2008;1:4–5.
[3] Izquierdo J, López P, Martínez F, Pérez R. Fault detection in water supply systems using hybrid (theory and data-driven) modelling. Math Comput Modelling 2007;46(3):341–50.
[4] Yoon S, Ye W, Heidemann J, Littlefield B, Shahabi C. SWATS: Wireless sensor networks for steamflood and waterflood pipeline monitoring. IEEE Netw 2011;25(1). http://dx.doi.org/10.1109/MNET.2011.5687953.
[5] Stoianov I, Nachman L, Madden S, Tokmouline T, Csail M. PIPENET: A wireless sensor network for pipeline monitoring. In: Information processing in sensor networks, 2007. IPSN 2007. 6th international symposium on; 2007. p. 264–73.
[6] Wu D, Youcef-Toumi K, Mekid S, Ben-Mansour R. Wireless communication systems for underground pipe inspection. Report, King Fahd University of Petroleum and Minerals Massachusetts Institute of Technology; 2017.
[7] Chatzigeorgiou DM, Youcef-Toumi K, Khalifa AE, Ben-Mansour R. Analysis and design of an in-pipe system for water leak detection. In: ASME 2011 international design engineering technical conferences and computers and information in engineering conference. American Society of Mechanical Engineers; 2011, p. 1007–16.
[8] Min Lin IW. Wireless sensor network: water distribution monitoring system. In: 7th IEEE radio and wireless symposium; January 2008.
[9] Hussein A, El-Nakib A, Kishk S. Energy-efficient linear wireless sensor networks applications in pipelines monitoring and control. Energy 2017;1:6.
[10] Zhu J, Li X, Zheng W. Design of early warning system for underground pipeline safety based on wireless sensor network. In: Chinese Automat. Congr.. IEEE; 2017, p. 95–8.
[11] Babazadeh M, Kartakis S, McCann JA. Highly-distributed sensor processing using IoT for critical infrastructure monitoring. In: Proceedings of APSIPA annual summit and conference. APSIPA ASC; 2017, p. 1–10.
[12] Kartakis S, McCann JA. Real-time edge analytics for cyber physical systems using compression rates. ICAC 2014;14:153–9.
[13] Arduino. Arduino101. 2016, http://docs.zephyrproject.org/boards/x86/arduino_101/doc/board.html.
[14] SEMTECH L. What is lora/technology. 2017, https://www.semtech.com/technology/lora/what-is-lora.
[15] TURCK. Water pressure sensor. 2017, https://files.clrwtr.com/userfiles/ct/documents/turck/turck-pt4400-pressure-sensors.pdf.
[16] Oberhumer MF. Open source lzo. 2016, http://www.oberhumer.com/opensource/lzo/.
[17] Kraus J, Bubla V. Optimal methods for data storage in performance measuring and monitoring devices. In: Proceedings of electronic power engineering conference 2008. EPE; 2008, p. 1–3.
[18] Kartakis S, Yu W, Akhavan R, McCann JA. Adaptive edge analytics for distributed networked control of water systems. In: Internet-of-things design and implementation (IoTDI), 2016 IEEE first international conference on; 2016. p. 72–82.
[19] ePHOTOzine. Top 12 best sd memory cards tested. 2016;2016(Dec 26, 2016). https://www.ephotozine.com/article/top-10-best-sd-memory-cards-tested-2016-17827.
[20] Adelantado F, Vilajosana X, Tuset-Peiro P, Martinez B, Melia-Segui J, Watteyne T. Understanding the limits of LoRaWAN. IEEE Commun Mag 2017;55(9):34–40.
[21] Kartakis S, Choudhary BD, Gluhak AD, Lambrinos L, McCann JA. Demystifying low-power wide-area communications for city IoT applications. In: Proceedings of the tenth ACM international workshop on wireless network testbeds, experimental evaluation, and characterization. ACM; 2016, p. 2–8.