Highly-Distributed Sensor Processing using IoT for Critical Infrastructure Monitoring

Mehrdad Babazadeh*, Sokratis Kartakis† and Julie A McCann†
* Imperial College London, London, UK and University of Zanjan, Iran
E-mail: mebab@imperial.ac.uk and mebab@znu.ac.ir
† Imperial College London, London, UK
E-mails:{s.kartakis13, j.mccann}@imperial.ac.uk

***Abstract*—Highly-distributed signal processing for critical monitoring infrastructures has been a main research topic over the last decade. Under this context, we show the three phases of the joint "Cyber-physical control system" project; a collaboration between Imperial College London and NEC Corp. Japan. First, the implementation of edge processing with multiple tasks including data mining and communication, developed on a lightweight single core low-powered MCU system is presented. This algorithm has been effectively customized to be implemented on resource-constrained embedded systems. The developed sensor network is coupled with a low-powered wide range LoRa platform for transmission of the minimized payload. The work explores the node-to-node communication limitations and discusses how edge processing can be used for water network control and we present the overview of a Cyber-physical control system which is concerned with the event-triggered control of a water network. Finally, the results of the LoRa communication tests are given.**

I. Introduction

Next generation smart cities, manufacturing, smart buildings, driver-less vehicles, precision agriculture etc. will go beyond sophisticated telemetry and will enable automatic distributed control and monitoring. That is, we will move beyond merely understanding people, places, and things via sensing and analysis to being able to close the loop and provide more automation. However, there have been few works on understanding how to build high performing computer systems composed of low-cost and low-powered wireless networked components that will support such systems. Though such systems are well motivated in theory to provide cost-effective adaptive control, traditional control systems users currently do not trust them as they do not provide any guarantees regarding stability. At the same time suppliers of traditional computing systems do not fully understand the behaviors exhibited by embedded networked systems, much less a control system. This has motivated a new research discipline focusing on the nature of cyber-physical systems and their interaction which has the potential to open up massive new markets.

One of the most important applications, the development of wireless sensor network (WSN) platforms to support critical, potentially inherently vulnerable and expensive infrastructures such as water networks is gaining interest. Their use brings about the ability to monitor utilities to provide early warning for deterioration or failure (e.g. leakage). Within the NEC project, Cyber-physical Systems (CPS) control project, we develop the integrated self-adaptive protocols that support distributed near real-time control and monitoring and provide guarantee pertaining to reliability, stability, convergence, and security. The extreme complexity of analytics required to process a huge volume of data limits what can be understood about the systems in real-time. Yet, real-time analytics is what is required to enable automatic control and monitoring. To overcome this problem, our approach is to reduce the dimensionality by minimizing the data without losing information. In "Cyber-physical control system" project, we have achieved this goal by focusing in compression, anomaly detection and control. This paper represents the three phases of the project and focuses on the implementation of the anomaly detection on a single-core platform. In addition, Section II illustrates a pseudo code for a future dual-core implementation. Section III briefly addresses event-triggered control approach. Since the proposed distributed system uses a wide area communication platform, LoRaWAN, we summarize our preliminary experimental results by evaluating state-of-the-art Low-Power Wide-Area (LPWA) technologies in Section IV.

II. Anomaly detection

There have been some research works in the area of water networks anomaly detection where they have applied the detection system on a wired network, a supervisory control and data acquisition (SCADA) system in [1] and other research work in [2] where they refer to the water network infrastructures and modeling. Researcher in [3] introduces a WSN to detect and identify major anomalies in steam flood pipeline networks. Another article for the field of water pipeline anomaly detection using sensor networks is [4] which describes the use 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 basically what many subsequent researchers have followed since. However, the major disadvantage of this approach is that all sensed or aggregated data is expected to be sent to the backend system for analysis which means that battery powered nodes will deplete their resources as they have to communicate all the data to the server and in some cases, this is not completely necessary. The main concern of this phase of NEC project is to detect anomalies based on the

water pressure in a water network by using a WSN and to transmit the corresponding data in two stages: First to transmit a short notification to the center including information about the sensor identity and the time of detection. Then, to transmit full compressed data of (before/during/after) anomaly.

This section translates the above-mentioned requirements to an implementable scenario of the anomaly detection by developing an Arduino-based WSN as follows:

- The fastest possible reading of analog sensor (Pressure sensor). The first candidate sampling time for sensor reading occurred every 2 ms. However, to have several time-consuming operations in the main loop of the program as it will be discussed later in this section, we had to increase it to 5 ms. Otherwise, with lower sampling rates, input data during the loop execution time could be overwritten which is not acceptable.
- Edge data analysis by:
  - Using a lossless compression algorithm to obtain a data Compression Rate (CR) for the rest of the algorithm as mentioned in [5]. This data can be decompressed for further analysis later in the center.
  - managing a SD card read/write procedure to record compressed data arrays and use it.
- Appropriate filtering to improve the performance.
- Anomaly Detection based on the CR.
- Real clock time tracking and synchronization.
- Node-to-node communication.

The above-mentioned tasks are different in terms of hardware and software requirements; and in how they interact with each other. Put simply, reading sensor data requires fast operations to be carried out every few milliseconds, while writing/reading to/from an SD card and communications are tasks that run slower which have to be taken into account.

A. Water network and embedded system architecture

Fig. 1 briefly illustrates the proposed architecture covering a section of a Water Network equipped with a WSN (with tens of SNs located in a 3-5km circle area), a base station where further data evaluations are carried out, and the cloud to keep data available through the Internet. As an example in a part of the network represented by the red color, some of the local SNs (SN1, SN3, and SN4) can sense the anomaly, a water leakage or burst in the pipes or other equipment. As represented in Fig. 2, anomaly detection system is including

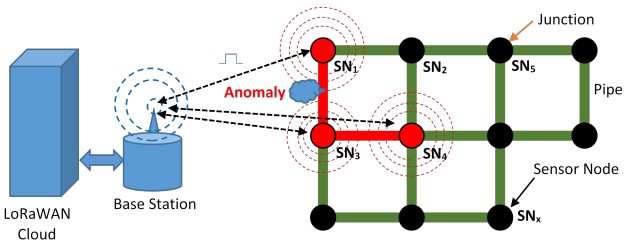

Fig. 1. Water Network architecture with SNs, Base Station and Cloud

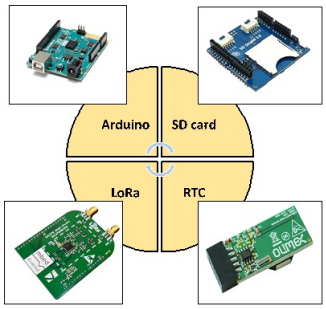

Fig. 2. Hardware platform and peripherals.

four main devices: An Arduino 101 board [6] where the main program is coded both for the measuring SN and the central node, an SD memory card where the compressed data is recorded together with the relevant timestamps and a special read/write operation is carried out, a Real Time Clock (RTC) to provide the real clock time for the system, and a LoRa shield SX1276 for the communication purposes.

Although by using an Arduino 101 we are constrained by the SRAM limitations (24 KB), there are reasons we chose it among the other available processors. The methodology optimized for the limited resource, single core, but low power Arduino 101 will obviously work with more powerful processors. There are also several peripherals compatible with this processor and useful libraries are available which makes the complicated implementation easier. The last reason is that the final implementation will be on a new upcoming dual-core Intel processor from the same platform but with Zephyr operating system. It will be including LoRa shield and more RAM space (80 KB) which makes it a perfect base for the proposed system after it is optimized on Arduino 101.

B. Single core implementation

Algorithm 1 and Algorithm 2 in the next page represent the pseudo codes for the main loop (thread 1) and the sensor reading unit (thread 2) in the Arduino-based anomaly detection system respectively. After the initialization of the process, the first task to do is to frequently read the input sensor and scale the measured values between 0 and 255 to be represented by a one-byte unsigned char variable. To get the best from the data we sample at the maximal sample rate achievable for the sensor and node system we can handle. To prepare the noiseless data for the process, a pre-filter applies and then the filtered data is recorded in an input array. The Pre-filter improves the data compression performance which in turn reduces the payload for the slow LoRa based communication [7]. This filter can be more accurately tuned later according to the given water-pressure data. Fig. 3 represents raw and the filtered signals by using either a first order low pass or a moving average filter (with 4 last data considered). The low-pass filter in the Laplace form is simply written as:

$$\frac{Y(S)}{U(S)} = \frac{1}{(a \times S + 1)} \qquad (1)$$

The discrete time representation of the above filter based on the Backward Euler method is given by (2) where α and T_s are the filter time constant and the sampling time.

$$S = \frac{Z-1}{Z \times T_s} \; ; \; y_t = \frac{a \times y_{t-1} + u_t \times T_s}{a + T_s} \qquad (2)$$

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 Compression Rate (CR);
  Apply Kalman Filter to CR;
  if *received any Synchronization Request* **then**
    Update RTC;
  else
    Read RTC;
  end
  Time stamp= RTC time;
  if *there is Anomaly or it is inside timer1 period* **then**
    Start timer1 by the first anomaly;
    Count time stamps until timer1=1 min ;
    Look for the worst anomaly during 1 min period;
    Notify Center at the end of 1 min period;
    Reset timer1 and Start timer2;
  end
  Append time stamp 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 anomaly detection

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

Algorithm 2: Pseudo Code for Callback function

A wait state in the first stage of the loop ensures filling up the input array with the filtered sensor data that pipelines that data to the next important stage (compression algorithm). It is noteworthy that even during the main loop, there will be several interrupts to read new sensor data. After the input buffer is full, it's time to get the real timestamp for the data set. The pre-filtered data is stored in an array then compressed and then the CR% is calculated and edge-anomaly detection is carried out.

C. Real time clock

A sensor node should communicate a short packet after an anomaly is detected together with a time-stamp which is just the indicator time. the real time-stamps can be provided when the extra data is decompressed back in the central node and will enable more accurate analysis. To retrieve the real clock time in the system and to synchronize all the SNs, a real-time clock (RTC) element is added to each node. It makes the real time available wherever it is called in this sequential implementation of the tasks. In this regard, an I^2c based, Mod-RTC Olimex has been used and programmed to provide required calendar (Year-Month-Day-Hour-Min-Second). For example, 170310141023 represents the time stamp of the compressed data packet on the 10th of March 2017 at 14:10:23. The sequential implementation forces the system to interact according to the time scheduled for each task located in the queue. This will affect the resolution of the timers in the system. For example, since we call the RTC every 4 seconds in the loop, we have to accept 0-4 seconds tolerance for the timer. After the first anomaly is detected, the proposed timer waits to react after the pre-defined 60 seconds (60-64 in practice). Then, the system starts notifying the base station and to refresh the anomaly detection procedure and then starts to analyze the possible anomalies for the next period. It restarts another 60 sec., waiting for the data request from the base station. Each SN involved in the anomaly detection makes a short packet including a time stamp provided by the RTC. This unique time points out to the time when the compression rate is going to be filtered and the anomaly is detected afterward. Another application of the RTC in this WSN is for the synchronization of the SNs whenever the real clock time is broadcasted by the base station. This might be needed for instance, every one month depending on the RTC accuracy or the other system requirements. The synchronization unit in each SN is located in the middle of the slow main loop and the SNs work independently. Therefore, the real time should be available when each SN checks out the receiving packets every 4 sec. Therefore, synchronization packets transmitted by the center has to be refreshed frequently for at least 4 seconds to make sure all SNs get the right clock time and set themselves properly and on time.

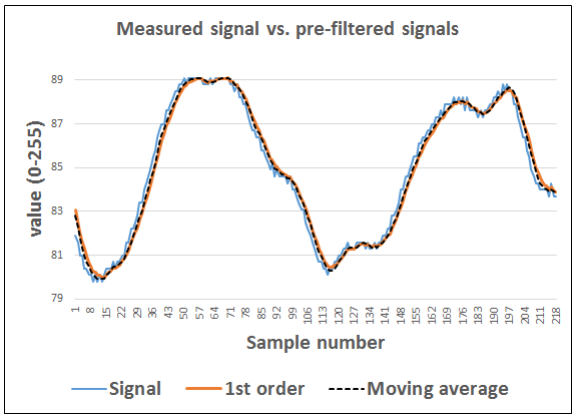

Fig. 3. Real measurement vs. filtered signal.

D. Lossless compression algorithm and signal conditioning

Following the methodology proposed in the previous stage of this project, in the Smart Water Lab project [5]. An anomaly is detected based on the variation of the CR instead of the raw data and the compressed data is recorded and communicated instead of the raw data. The CR is defined in (3) where the 'in_len' and the 'out_len' are input and output array sizes.

$$CR(\%) = \frac{in\_len - out\_len}{in\_len} \times 100 \quad (3)$$

The compression technique used for this first version is lossless and therefore data quality is maintained. To do this, the implementation of the lossless Lempel Ziv compression algorithm is the vital part for the whole anomaly detection system. Because of the low RAM memory available in Arduino 101, even the minimal version, Mini-LZO, which is developed by Oberhumer as mentioned in [8] and [9] was not applicable due to the RAM space needed though it was already used in [10] on Intel Edison platform. Another library of LZO so-called LZO1X_1_11 was customized and could work on Arduino 101 after manipulating the dependent libraries and changing memory space allocation. This library needed a predefined memory allocation for the input and output arrays where the maximum input array size could reach to 790 bytes after optimization of the memory usage of the different variables and arrays. The length of the input array of the data compression algorithm keeps fixed and known (790 Bytes) where the output length is known but not fixed. A good compression (CR% greater than 90 %) achieves for an input data window with the minimum data variation and a poor compression (lower CR%) represents more variations among the taken input-data. After a single dimensional Kalman filter [11]-[12] applies to the CR to minimize false positive anomaly detection, the filtered CR is evaluated as seen in Fig. 4. The initial value of the filter output is also set to 100 to avoid false anomaly detection while initiating the system.

A great measured signal variation can be reflected also in the CR and by using a proper threshold a miss behaving is detected. In addition, the severe and repetitive faults versus single faults should be taken in to account where they result in to repetitive event messages issued by a SN. As a solution, only the worst anomaly during a specific period of time, for instance, 1 minute of system operation is transmitted together with the corresponding time-stamp. The system then refreshes the worst anomaly detection procedure and re-starts to monitor anomalies for the next 1 minute afterward.

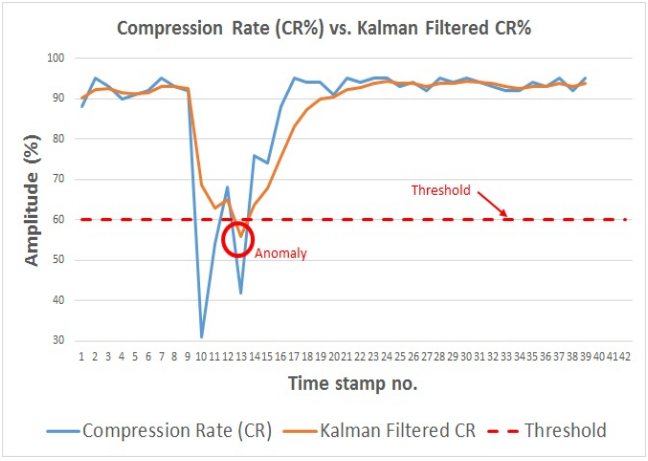

Fig. 4. Data compression Rate before and after using Kalman Filter.

E. SD Card read/write

Because of the Arduino's memory limitation, we use an SD memory card for the data read/write purposes. After the anomaly detection task, the compressed data arrays are written together with their corresponding time-stamp to the SD card. To writing/reading the long and variable-size data packets to/from SD card, Algorithm 3 and Algorithm 4 present the algorithms developed for a stackable SD/TF Card Shield [13].

Data: Address of the last written data.
Result: A ppend a compressed data packet to a 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 Time Stamp ;
  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

Data: Required history (time) of the recorded data.
Result: Make a data string of one compressed data packet together with the time stamp to transmit.
Open text file on the SD memory card;
if *not reached to the end of the required packet* **then**
  if *starting point has not already been found* **then**
    while *until reaching to the required address* **do**
      Go one step back in memory address;
    end
  end
  1. Start reading until the next time stamp;
  2. Make a data string of (1 packet and a time stamp);
  if *node-to-node* **then**
    Transmit pieces of a packet one by one in a loop.
  else
    Transmit the string to the base station;
  end
end
Close file;

Algorithm 4: Reading SD card and transmission

F. Optimal input array size

Fig. 5 shows a simplified main loop of the program for the anomaly detection in the SN-side. The input array size is an important parameter which has to be set correctly by the user. It affects almost every task in the main loop and causes a system failure if it is wrongly chosen. The following conditions apply for an optimal input array size selection.

Increasing the input array size for the slow tasks in the main loop increases t1 and the total RAM needed. By using a minimum possible data format, unsigned char to assign only one byte for each number, we guaranty to use the lowest possible RAM because input data format affects other dependent variables accordingly and influences the total memory allocation. Further, increasing the input array size increases executing time of the wait loop and makes the following advantages (ADV) and disadvantages (DIS):

- (ADV): Enhances the performance of the data compression algorithm and reduces both the memory needed for the data logging and communication per specific input length.
- (DIS): Depends on the type of data communication, LoRa might fail to transmit the larger compressed data arrays due to the inherent payload limitation. Further, the system loses the real-time sense of the anomaly detection since it has to wait a long time until the long input array is ready for the compression and anomaly detection.

On the other hand, reducing the input array lowers t1 and it causes:

- (ADV): Improves the real-time sense of the anomaly detection so that anomalies are detected after a short time they occur. Further, since maximum compressed data array size is limited to the input size, the communication time is limited accordingly which is good for the node to base station communication platform.
- (DIS): There will be a worse data compression and the data logging performance. This means longer compressed data is needed to be written on the SD card and finally, a longer total communication for the whole data in comparison to the case we could take longer inputs is expected. Finally, if $(t1 < t2)$, data are overwritten in the input array before being compressed.

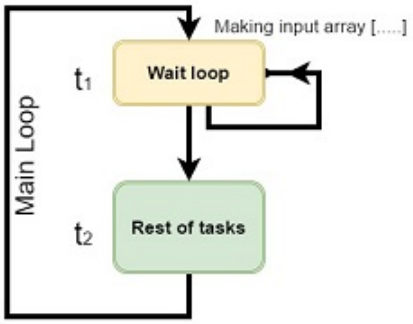

Fig. 5. Simplified flowchart of the sequentially implemented main loop.

G. Anomaly detection on a dual-core processor

In the single core Arduino-based system, we reached to maximum 790 bytes input data and had to increase the sampling time to 5 ms to achieve a safe operation as already proposed by t1>t2. Then the required time for filling up the input array became t1= 790 * 0.5 ms $\approx$ 4 sec which is the maximum permitted time for t2. If we are able to increase the number of input array data, we will be able to allow more time for the rest of the tasks. This can be done even by a processor with a more RAM space, regardless of being the single core or dual-core.

By a new Intel Processor (SE C1000) with more RAM available, not only the input array size can be increased, but also the sampling time of the sensor measurements can be minimized. This lowers the whole data size for a specific period of time which results in a lower total payload for Communication. Algorithm 5 presents how the proposed implementation would be systematically ported to a dual-core system.

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 Compression Rate (CR);
  Apply Kalman Filter to CR;
  if *received any Synchronization Request* **then**
    Update RTC;
  else
    Read RTC;
  end
  Time stamp= RTC time;
  if *there is Anomaly or it is inside timer1 period* **then**
    Start timer1 by the first anomaly;
    Count time stamps until timer1=1 min ;
    Look for the worst anomaly during 1 min period;
    Notify Center at the end of 1 min period;
    Reset timer1 and Start timer2;
  end
  Append time stamp and data packet to SD memory card (refer to Algorithm 3);
  if *There is a data request and* $T_{timer2} \leq 1$ *min* **then**
    Read SD card (Algorithm 4) and pass data to core 2 of dual-core processor for communication;
    Don't wait for core 2 and continue loop execution;
    Reset timer2;
  end
end

Algorithm 5: Pseudo code for anomaly detection implementation on a dual-core processor

In this implementation, communication tasks can be done by the second core and the main loop doesn't need to wait for the execution of the slow communication units. In addition, to

take advantages of using more RAM space, this can introduce great advantages to the system performance by either of the following ways:

- Taking out the time-consuming communication from the main loop saves time for the rest of the tasks.
- More RAM space allows using the longer input data size for the compression algorithm. The longer input array, the longer time for filling up the input array. But, we can reduce the sampling time to overcome this issue. At the same time, we take advantage of the proposed saved time due to use the second core. Then we will have a better compression performance and lower SD card memory space required to record the data in comparison to the single core platform.

H. Timing in the implemented system

For the sake of simplicity of description, Fig. 6 in the next page represents a graphical overview of the implemented anomaly detection. Some of the diagrams in this figure are time-based and the rest show the sequence of executing the different tasks. The top diagram (A) denotes to a pre-filtered measured signal which will be the source for the rest of the procedure. The second diagram (B) represents the sequence of the real time fixed-length arrays of filtered data which are used to make inputs for the data compression algorithm. The next diagram (C) shows the lengths of the compressed data arrays. Defined later in (3), the data compression rate (CR) is obtained by using corresponding values shown in diagrams B (input length) and C (output length) and a filtered version of CR (FCR) is represented in diagram (D). After the first anomaly is detected by comparing the FCR to a fixed pre-defined threshold, the worst anomaly (smallest FCR) is selected among the other possible anomalies in a period of 1 min. Then, the SN transmits the worst anomaly information to the center and the center is given 1 min deadline onwards to send its data request. Diagram (E) indicates the compressed data packets that are appended sequentially to the SD memory card data. After receiving data request shown in diagram (F), the SN looks back to its last 2 min data packets in the SD memory card as represented in the diagram (G) and starts reading appropriate data and then making a string and then sending it to the center packet by packet, according to the LoRa payload limitation.

As mentioned, anomaly detection program includes a main loop with different tasks (included in thread 1) and a callback function (thread 2) which is called when an interrupt service routine takes place by the thread 1. Since the SN-side tasks execute consecutively with the interrupt/s in between, it is important to consider their individual and total required time. The number of interrupts during the loop execution depends on the maximum time required for each task in the loop and also the defined sensor sampling time (interrupt time). Therefore, the sampling time can be minimized only after all tasks are implemented and tested in the worst cases. Analyzing the timing tests in this section resulted in the minimum sampling time in this application which is $T_s = 5$ ms where a lower sampling time may arise a risk of data loss. The time required for each individual task in the worst case scenario, where there is no data compression at all, is represented in Fig. 7. In the earliest stage of this diagram, an internal wait loop that executes in Δt_1 milliseconds ensures that a maximum possible, fixed size input array (N=790) is filled up before data compression algorithm starts. The wait loop duration depends on the previous loop execution time. The wait loop will work as expected provided that the total time for executing tasks is less than the time required for filling up the input array. Further, $\max(\Delta t_1)$ refers to the maximum total time needed for filling the input array up with 790 numbers as:

$$\begin{aligned}\max(\Delta t_1) &\approx \textit{Input array length} \times T_s \\ &= 790 \times 5\ ms = 3950\ ms \approx 4\ sec \end{aligned} \quad (4)$$

Therefore, to avoid data to be overwritten in the input array, following relation must be fulfilled where $T_{task(n)}$ is execution time of the 'nth' task:

$$\sum_{n=2}^{8} T_{task(n)} < \max(\Delta t_1) = 4\ sec \quad (5)$$

Fulfilling inequality in (5), the main loop will run in an approximately fixed time about 4 sec. The performance of the writing and reading tasks, represented in Fig. 7, are discussed in the following:

1) Data writing performance: The writing to the SD card is the third slowest task in the present implementation and it takes $\Delta t_{6(\mathrm{Max})} = 153$ ms to write the longest possible packet. Obviously, we cannot expect the same occupied space (790 bytes) after it is written. It is instead:

$$790\ \textit{unsigned char data} \equiv 3591\ \textit{bytes} \approx 3.5\ \textit{KB} \quad (6)$$

This means that due to having metadata which is written together with each individual unsigned char data, the text file size is about 5 times bigger. Therefore, the overall writing speed (S_W) to SD memory card can be obtained as:

$$S_W = 3591/(1024 \times 153 \times 10^{-3}) = 23\ KB/s \quad (7)$$

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 advance SD-Fat library may help to speed up the writing procedure, it will need more SRAM which is not available in the Arduino 101. Theoretically, every 3.5 KB data is written in each loop time (about 4 sec) in the proposed worst case scenario. Therefore, one minute of data should be equal to 52.5 KB in the SD memory card. However, measurements show that after one minute, 72 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 162 days as calculated in the beneath:

$$t_{fill(min)} > (16\ GB)/(72\ KB/min) \equiv 162\ days \quad (8)$$

Where $t_{fill(min)}$ refers to the minimum time needed to fill up the SD memory card from the first line to the end. This

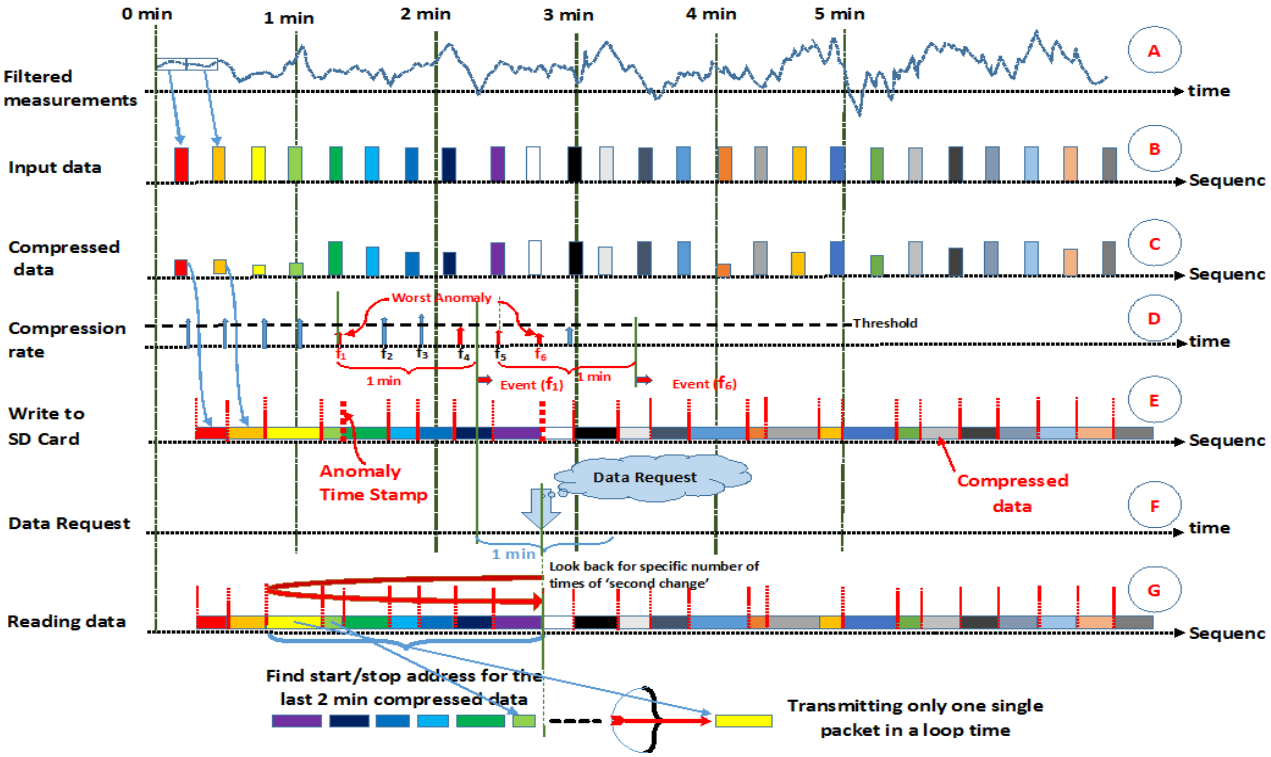

Fig. 6. Operational and timing diagram of anomaly detection

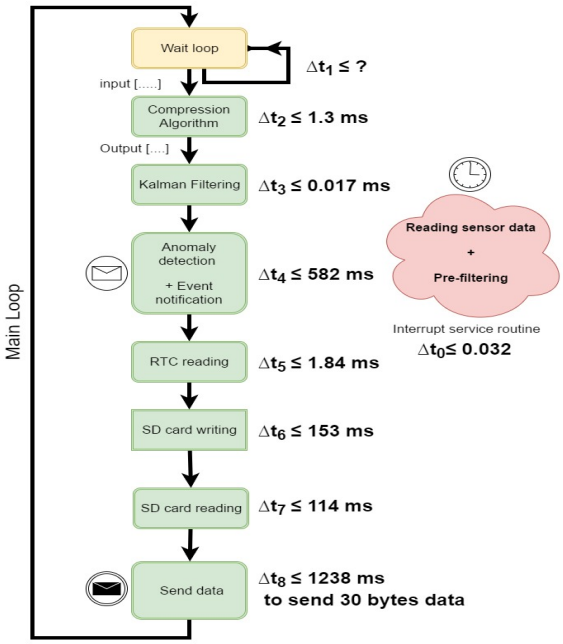

Fig. 7. Flow diagram of the maximum times consumed by different tasks.

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_{fill} we will have. It is because the output (compressed data array) size will be smaller than 790 for a fixed loop time and will take longer until SD card is filled up.

2) Data reading performance: The packet size of 790 unsigned char numbers is different than the text file size of these numbers when they are seen in the SD card text file. Therefore, two values for 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 (TR_1) of the whole system including SD memory card. Every 790 unsigned char numbers are equal to 3591 bytes on the SD card and the corresponding time to read and make a string of the data packet is about 114 ms. Therefore:

$$TR_1 = 3591/[1024 \times 114 \times 10^{-3}] = 30.7\ KB/s \qquad (9)$$

The second value gives the effective data transmission speed (TR_2) according to the pure data size as follows:

$$TR_2 = 790/[1024 \times 114 \times 10^{-3}] = 6.7\ KB/s \qquad (10)$$

The required time of the reading task, Δt_7 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. Finding the starting point, it reads one packet of data and makes a data string. 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. The next sections address the other phases of the NEC project

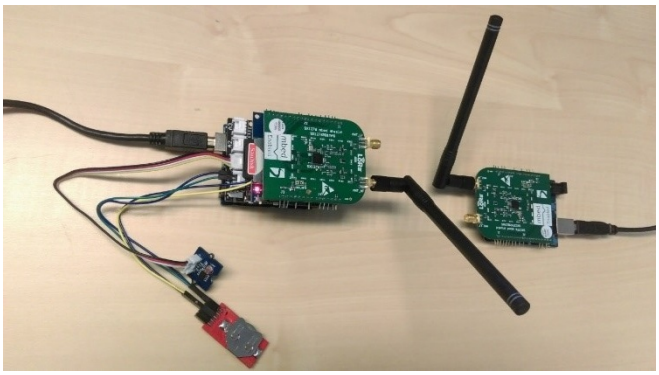

Fig. 8. Arduino based WSN with LoRa module SX1276 for communication.

where the event-triggered control part and communication test have been carried out.

I. LoRa based node-to-node data Communication

The Sensor Node (left hand side device) and the central node (right hand side device) in Fig. 8 use the LoRa module, SX1276 for communication. The SN-side hardware needs more devices connected and more complicated code to manage different tasks in comparison to the central node. The SN-side is developed by Arduino 101, LoRa module SX1276, SD Card Shield, RTC module, analog extension board, and a light Sensor. However, the center node side is developed by only an Arduino 101, and a LoRa module SX1276. Communication between the SN-side and the central node is carried out in different stages as follows:

- For synchronization of the SNs based on the message received from centre. Hear is an example of the short message (S1170627161500) which is updated according to the real clock time. From the left-hand side, the first letter 'S' is a symbol for synchronization and the second letter represents the unique number of the central node '1'. Then, Year (2 digits), month, day, hour, minute and second come afterward.
- On detection of the anomaly, only the originating node(s) will send notification immediately to the central node. Nodes not seeing the event remain silent. This data indicates the source node and the time detected. A carrier sensed MAC protocol is used for this notification so that there is no delay in sending data because the nodes are sparse and this means that interference should be low, so this approach minimizes delays indicating an anomaly.
- After the central node receives anomaly notifications, it then asks for extra data from the source nodes only with a schedule indicating when they can send data back to the central node.
- On receiving a data request from the central node, the source SNs then send their long time compressed data arrays for a long while during their several loop times.

The experiments done during this research represents that we cannot send more than 31 bytes of data together with a time-stamp in a node-to-node LoRa platform even for the highest LoRa mode 10. However, the compressed data packet size will always be more than this length. Therefore, we need to divide long compressed data packets to the fixed but short packets possible for transmission. Therefore, we have to send the long data as follows: Primarily to transmit the timestamp and then to divide any long compressed data array to several 30 bytes long packets and transmit sequentially in every SN-side loop execution. Due to a fixed input array size assigned for the compression algorithm (In_len), the range of the compressed data (Out_len) is known and smaller than input array. We know In_len = 790 bytes (fixed) and Out_len $<$ 790 bytes (variable).

While writing to the SD card, the memory limitation applies and we stop to write when the size of the last compressed data packet for writing on the SD card is greater than a pre-defined value one kilobyte less than SD card memory size (for example 16 GB). After this point, we remove all the long-time data from SD card and write newly compressed data arrays starting from address zero (0).

J. Validation of the SN-side anomaly detection

To describe how the system is detecting anomalies, we have intentional applied repetitive anomalies by changing the sensor measurement drastically for a while. It changes the filtered CR as represented in Fig. 9. between packet number 3 to 7. The system works according to the threshold for CR, set to 85%, and Table I illustrates how the anomaly detection system behaves.

For the sake of simplicity in description, we only take last six digits of every timestamps in Table I to represent: hour, minute and second for events. The system has been set to take the last 20 seconds (linked to timestamps 064139 to 064159) of compressed data including (before/during/after) anomaly for the data communication. During this experiment, the SN detects the worst anomaly (occurred at 064139) and transmit it to the central node (at 064151) which is the end of the waiting period. After that, all the required data during nine loop's operating times which takes about 30 seconds

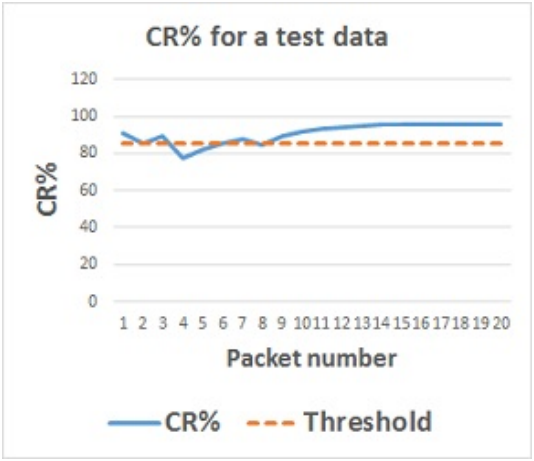

Fig. 9. CR% vs. packet number

TABLE I
ANOMALY DETECTION OUTPUT

CR%	Time-Stamp	System status
90.54	3-170626:064127	System is working normal
85.25	3-170626:064131	System is working normal
89.06	3-170626:064135	System is working normal
77.32	3-170626:064139	First anomaly detected
81.88	3-170626:064143	Second anomaly detected
85.05	3-170626:064147	System is working normal
87.37	3-170626:064151	Notify F3:030403064139
84.50	3-170626:064155	Full data starts to be sent
88.99	3-170626:064159	send the rest of data
91.72	3-170626:064203	send the rest of data
93.39	3-170626:064207	send the rest of data
94.41	3-170626:064210	send the rest of data
95.03	3-170626:064214	send the rest of data
95.41	3-170626:064218	send the rest of data
95.64	3-170626:064222	send the rest of data
95.78	3-170626:064226	send the rest of data
95.87	3-170626:064230	send the rest of data
95.92	3-170626:064234	back to normal
95.95	3-170626:064238	working normal
95.97	3-170626:064242	working normal

in this test is transmitted to the centre. After finishing the data communication, system represents a normal situation (at 064234) where there is no threshold violation. The first column in this table represents the filtered compression rate (%) which is the criteria for the anomaly detection. The second column starts with the sensor number that in this case is '3' and continues with the whole time-stamp. The third column shows the system status which can be either of the normal, during fault period, or during communication.

III. EVENT-TRIGGERED CONTROL AND COMMUNICATION OPTIMIZATION IN A TESTBED ENVIRONMENT

Automated water distribution networks are an open research field for the wireless sensor and actuator network (WSAN) and CPS. This research aimed to augment the traditional water supply network by using WSN/WSAN and CPS technologies to satisfy the real needs of the water companies and to define new innovative directions under the context of Smart Cities. The objective is to design an innovative Wireless Sensor/ Actuator Network communication protocol that ensures the stability and real-time control of an automatic Water Distribution System. This system has to ensure the stability of both the wireless communication and water network at the same time. The control part was done in three steps: First, a small-scale testbed was created as a simulator mimicking a real water network. The deployment of classic and event-trigger control theory allowed the optimal decision of sample rates and precision of the measurements to be measured therein. The deployment requires the mathematic modeling of the testbed, stability, virtual simulation, and the development of algorithms in real nodes. Then, the decentralization of event-triggered control and definition of the trade-offs between stability and communication network lifetime were studied. After the completion of the testbed and the deployment of the control theory approach, the next step was the augmentation of event-trigger control theory, in order to involve in the threshold defining the status of each sensor/actuator node. The main goal was to define the trade-off between the stability and the lifetime of the system. Finally, the On-line adaptation of the event-triggered system in faulty nodes was explored. The decentralized event-trigger system was incorporated with our anomaly detection algorithm to predict the failure of sensor nodes, and fill the data gap by adjusting the sample rate of the correlated nodes in order to estimate the missing data. This step requires the use of correlation algorithms combined with the event-triggered threshold definition obtained in the previous phase. The main concerns of this phase were the system scalability, optimality, the extension of system lifetime, memory efficiency, adaptation, and robustness. Our first experiments to fulfill these concerns were on energy consumption reduction (data fusion, aggregation, lossy, and lossless algorithm), hardware limitations and improvements, wireless communication optimization, and in-node data analysis. The results of these experiments were data fusion and compression algorithms comparison, the clear definition of the hardware changes, the time and energy constraints of 3G and Weightless communication, and the efficient in-node data-stream analysis algorithm for anomaly detection. Furthermore, the first outcomes of this research led to the publication of an innovative in-node self-adaptive anomaly detection algorithm, which requires 98% fewer computations than the traditional techniques [14].

IV. LPWA TECHNOLOGIES IN REAL IOT DEPLOYMENTS

Over the last decade, short-range wireless communication technologies, such as WiFi and Bluetooth, or cellular networks, such as GPRS and 4G, have been used to enable the monitoring of IoT infrastructure. However, the large scale of the real world IoT applications, either in terms of the number of sensor devices or the coverage, leads to unfordable costs. For example, by exploiting short range communication technology, an expensive gateway or a sensor node is necessary for every 10 to 20 meters. In the case of the cellular networks, the data fees per device increase dramatically, while the coverage is uncertain in remote areas. In order to enable

Fig. 10. BentoBox: Hybrid communication sensor node.

low-cost and city scale IoT applications, Low Power Wide Area (LPWA) wireless communication technologies have been introduced recently. Specifically, LPWA technologies exploit the unlicensed spectrum to transmit information in low power (e.g. 20mA) and frequency (e.g. 868MHz) with the trade-off of low data rates (e.g. 0.2 kbps). Under this context, two main theoretical directions have been used to enable the utilization of the spectrum: Spread Spectrum (SS) and Ultra-Narrow Band (UNB). In SS, the devices spread to signal along the spectrum under a certain bandwidth allowing the gateways to reconstruct the transmitted information reliably. In UNB, the devices transmit to a narrow bandwidth enabling million of devices and avoiding interference. State-of-the-art products of these technologies are LoRa by Semtech [15] for SS and Sigfox [16] for UNB. To evaluate the performance of LPWA technologies in real world IoT deployments based on device-to-device communication (D2D), we performed a number of experiments in [15] under different communication scenarios e.g. underground to overground, ground to ground, rooftop to ground. In these experiments, we used our hybrid communication sensor node, BentoBox (see Figure 10). BentoBox incorporates LoRa (spread spectrum), Xbee868 [18] (conventional narrow band) and NWave [17] (UNB) communication modules allowing a fair comparison among the technologies (more details can be found in [15]).

Overall, LoRa outperformed the other technologies in terms of range achieving 2.4 km in a semi line-of-sight environments within average 40% success rate (the longest distance with Xbee868 was 2km with 80% success rate). Similarly, in none line of sight scenarios, LoRa achieved the longest ranges at 850m, within average 70% success rate (while the range for Xbee868 and NWave was 460m and 70% and 20% the success rate respectively). However, LoRa dominated the other technologies in terms of the range, UNB promising a more efficient utilization of spectrum enabling more devices per gateway. Having the results of the above experiments, we currently examine the applicability of LPWA communication technologies in smart water networks. Under this context, we installed a number of BentoBoxes into chambers (i.e. underground) of the Welsh Water company water network, in Cardiff area. Preliminary results reveal that LoRa outperforms the other technologies. However, the ranges are reduced dramatically by achieving communication only within 315m. Our future work will be related to the type of antennas per application in order to achieve longer ranges.

V. Conclusions

This paper revealed details of an optimized multi-thread implementation of a new edge-based anomaly detection for the smart water networks on the Arduino 101. A portable lossless data compression library, LZO was customized to be used by the sensor nodes when they evaluate data compression rate in order to detect anomalies. On detection of anomalies by the sensor nodes, they provide time aware notification and compact data communication of pre-fault and post-fault for the central node with minimal payloads. The paper came up with different implemented and tested solutions and the time tests verified the system operation under the worst case scenarios and thereafter, resultant practical limitations were identified. The final part of the paper explored in brief how well current LPWA communications technologies can support the transfer of data from a pipe to above ground base stations.

Acknowledgment

Special thanks to NEC Corporation in Japan for funding this work as a part of the CPS-Ctrl project: Reliable Distributed Adaptive Control for next generation Cyber-Physical Systems. The authors would also like to show their appreciation to the members of AESE group for their valuable inputs.

References

[1] M Raciti, J Cucurull, S Nadjm-Tehrani, “Critical infrastructure protection,” *2012 - Springer* p. 98-119.

[2] J. Izquierdo, P.A. Lpez, F.J. Martnez, R. Prez, “Fault detection in water supply systems using hybrid (theory and data-driven) modeling,” *Mathematical and Computer Modelling*, Volume 46, Issues 34, August 2007, pp. 341-350.

[3] SunHee Yoon, Wei Ye, et all, “Wireless sensor networks for steamflood and waterflood pipeline monitoring,”*IEEE Network* (Volume: 25, Issue: 1, January-February 2011)

[4] I. Stoianov, L. Nachman, S. Madden, T. Tokmouline, and M. Csail, “Pipenet: A wireless sensor network for pipeline monitoring,” *in Proc*. IPSN, 2007, pp. 264-273.

[5] S. Kartakis, J. A McCann, “Real-time Edge Analytics for Cyber Physical Systems using data Compression Rates,” *Publisher: USENIX Association*, 2014, pp. 153-159.

[6] Arduino 101, https://www.arduino.cc/en/Main/ArduinoBoard101.

[7] LoRa Alliance, “LoRa Alliance Technology,”*https://www.lora-alliance.org/What-Is-LoRa/Technology* (accessed 1 December 2016).

[8] Markus F.X.J. Oberhumer. open source lzo. http://www.oberhumer.com/opensource/lzo/ (accessed 1 Dec. 2016).

[9] J. Kraus and V. Bubla, “Optimal methods for data storage in performance measuring and monitoring devices,”*in Proceedings of Electronic Power Engineering Conference*, 2008.

[10] S. Kartakis, W. Yu, R. Akhavan, and J. A. McCann, “Adaptive Edge Analytics for Distributed Networked Control of Water Systems,”*IEEE First International Conference on Internet-of-Things Design and Implementation*, 2016, pp. 72-82.

[11] Kristian Sloth Lauszus, “ practical approach to Kalman filter and how to implement it,”*http://blog.tkjelectronics.dk/2012/09/a-practical-approach-to-kalman-filter-and-how-to-implement-it/* (accessed 8 February 2017).

[12] TINKERING. Filtering Sensor Data with a Kalman Filter,”*http://interactive-matter*.eu/blog/2009/12/18/filtering-sensor-data-with-a-kalman-filter/ (accessed 1 January 2017).

[13] Stackable SD/TF Card Shield for Arduino V3. http://www.robotshop.com/en/stackable-sd-tf-card-shield-arduino-v3.html (accessed 5 January 2017).

[14] S. Kartakis, A. Fu, M. M. Espinosa, and J. A. McCann, “Evaluation of Decentralized Event-Triggered Control Strategies for Cyber-Physical Systems“ *arXiv preprint arXiv*, 1611.04366, 2016.

[15] S. Kartakis, B. D. Choudhary, A. D. Gluhak, L. Lambrinos, and J. A. McCann. Demystifying low-power wide-area communications for city IoT applications. ”*Tenth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization (pp. 2-8)*, 2016.

[16] Sigfox radio modules. https://www.sigfox.com/. (accessed 26 Jan. 2017).

[17] Nwave radio modules. http://www.nwave.io/. (accessed 26 Jan. 2017]).

[18] Digi, XBee-PRO 868, http://www.digi.com/products/xbee-rf-solutions/modules/xbee-pro-868. (accessed 5 Aug. 2016).