Adaptive Sampling for Saving Energy: A Case Study on The Libelium-based Environment Monitoring Systems

Phat PHAN-TRUNG
Faculty of Computer Networks & Communications
University of Information Technology - Vietnam National University - Ho Chi Minh City - VNUHCM
Ho Chi Minh, Viet Nam
phatpt@uit.edu.vn

Thuat NGUYEN-KHANH
Faculty of Computer Networks & Communications
University of Information Technology - Vietnam National University - Ho Chi Minh City - VNUHCM
Ho Chi Minh, Viet Nam
thuatnk@uit.edu.vn

Quan LE-TRUNG
Faculty of Computer Networks & Communications
University of Information Technology - Vietnam National University – Ho Chi Minh City - VNUHCM
Ho Chi Minh, Viet Nam
quanlt@uit.edu.vn

***Abstract*— In the plethora of energy saving techniques developed in Internet of Things, adaptive sampling is one of the common methods to reduce the energy consumption of IoT nodes, at the cost of reducing the data accuracy. Additionally, the user cannot define the amount of energy to be saved when performing the adaptive sampling technique. This paper shows a case study applied our developed UDASA – The User-Driven Adaptive Sampling Algorithm for Massive Internet of Things on the Libelium-based environment monitoring systems. The aim of this work is to support users to trade-off between energy consumption on IoT devices versus the data precision. The results show that once applied UDASA in 4 days, the collected data only takes about 10% compared to that of without UDASA, while the system saves 9% of energy, and the data accuracy is about 84% after interpolation.**

Keywords—UDASA, Internet of Things, IoT, Power consumption, Libelium

I. INTRODUCTION

The Internet of Things is of the hosted topics in recently years. The number of IoT devices connected to the Internet has already surpassed the number of humans on the planet. By 2025, the number of devices is expected to reach around 75.44 billion worldwide [1]. Figure 1 shows the number of IoT devices and Non-IoT devices from 2017 to 2025. This is a vast platform for IoT solutions. IoT is one of the core technologies of the 4th industrial revolution, beside Artificial Intelligence, Bigdata, and Cloud Computing. Besides the conveniences that

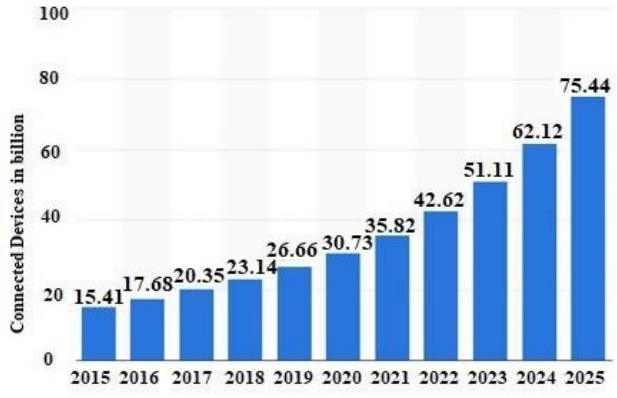

Fig. 1. IoT devices connect to the Internet from 2015 to 2025 [1]

IoT brings, there are still challenges that are being studied e.g., security and privacy, scalability, devices management, standard, as so as energy consumption.

Depending on applications running on its devices, the IoT node can always act and sense the environment data, then transfer it to IoT Gateway. Some IoT hardwares support IoT nodes change to sleep mode to reduce the energy consumption. Table I shows the comparsion of radio energy consumption of some IoT hardwares. When in active mode, IoT nodes use many times more energy than in sleep mode.

TABLE I. THE ENERGY CONSUMPTION OF SOME IOT DEVICES

Devices	Transmission Protocol	Radio chip	Active mode	Sleep Mode
TelosB	Zigbee	TI MSP430	1.8 mA	5.1µA
ESP8266	802.11n	Tensilica L106	56mA RX/ 120mA TX	15mA
Ra-02	LoRa	SX1278	12.5mA RX/ 93 mA TX	1.6 mA
Digi XBee® 3	NB-IoT	SARA-R410M-02B	190 mA	20µA

The periodic state change method is the simplest among IoT node energy saving methods. The IoT node only sleeps, then actives, senses and sends, then sleeps again. This process repeats periodically. To optimize this solution, the adaptive sampling is approached. The data transmission process is depended on data value. If the sensor data is stable, the sampling rate will be reduced to save energy. In contrast, if the sensor data fluctuates, the the sensing frequency will be increased to collect more data about current events.

To the best of our knowledge, at this time, no work applies the above approach into IoT physical devices. To address these challenges, the authors re-implement The User-Driven Adaptive Sampling Algorithm for Massive Internet of Things UDASA [2] into Libelium devices to evaluate this algorithm when running in real IoT devices. After re-implementing from its pseudo-code by python programming language, the authors evaluate this algorithm by 2 test case: one sensor with large data fluctuations like a light sensor, and three sensors have stable data like temperature, humidity, pressure.

Our key contributions are summarized as follows:

- Re-implement the UDASA into NOAA datasets to apply it to our specific scenarios.
- performs UDASA on our Smart Cities - UiTiOt datasets
- Take 2 scenarios: apply UDASA into one sensor and three sensors.

The rest of this paper is organized as follows: Section 2 lists and evaluates the work related to reducing the sampling frequency; Section 3 introduces our system architecture and the implementation of this system; In section 4, some scenarios are defined to evaluate UDASA when running in our system; This paper ends with our conclusion and discussion.

II. Background and Related Work

This section introduces the IoT device components, and the previous works related to energy saving on IoT devices. An IoT sensor device is combined from 4 components:

- Sensors and Actuators subsystem aim to sense environment data and take an action when receiving the control signal.
- Processing subsystem handles all operations of IoT node.
- Communication subsystem sends the IoT data collected from sensor to the applications and receives the control signal from users via the IoT gateway.
- A power source subsystem provides the energy for all other components of IoT nodes.

Besides the three first components, energy is also one of the research directions in IoTs. The related works on energy for IoT include optimize the schedule for sleep/active mode, change the threshold of sensing/sending, apply the role for sensor nodes. In sleep/active mode, the sensor devices are in sleep mode, by default. They only wake up when sense environment data and send it to the gateway. On the threshold setting case, the sampling can be set based on sensor node state, e.g., sensor node energy level, the sensor value features, the sensor connections, … In case 3, each sensor node is assigned a role, depending on the application running on the node, or on the location, power level of that node. Then, each sensor node is assigned a different frequency of sending sensor data.

Beside the fixed sampling, a new technical can be apply for IoT, namely Adaptive Sampling. This technique supports change the sampling frequency based on historical data and the level of energy savings that users want. This technique

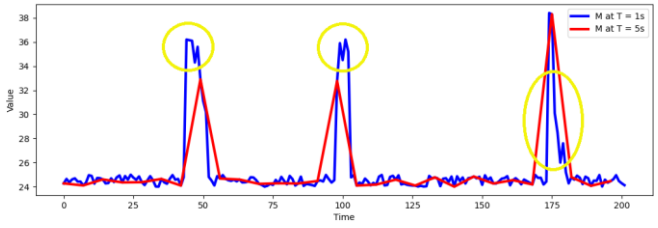

Fig. 2. The problem of periodic sampling

helps to trade off the amount of data collected and the level of energy savings. If the sensor data is stable, the sampling rate will be decreased for energy saving. In contrast, the sampling rate will be increased if sensor data has a high frequency of change.

According to [3], the sampling technique is cataloged as adaptive sampling and adaptive compression. Both of them aim to reduce resource consumption by changing the sampling rate of IoT nodes. The adaptive sampling's benefits are fewer sensor sends and sends, and higher precision. In contrast, adaptive compression sample signals below a certain rate and later reconstruct them with high accuracy. In this subsection, the authors evaluate some approaches related to adaptive sampling.

Trihinas et al [4] prensent AdaM – an open-source Adaptive Monitoring framework for IoT devices. The work incorporates two algorithms, e.g., adaptive sampling and adaptive filtering to correctly estimate the next event in the monitoring stream. The authors deploy this framework on Raspberry PI model B, Android Wear emulator and Fitbit Charge devices. At Result, AdaM reduces 74% data, while preserving a greater than 89% accuracy and energy consumption by 71%.

In [2], Le Kim-Hung et al proposed UDASA – an adaptive sampling algorithm based on the energy saving level that defined by user. This proposes trade-off between energy consumption and data precision. The authors of [1] evaluate this propose via NOAA dataset and IoT datasets. The result show that UDASA can reduce 20 times collected data compared with the traditional fixed-rate approach when the data accuracy is 96.55%.

The approach of B. Srbinovski et, al [5] is an adaptive sampling algorithm for energy-hungry sensors - an improvement, namely Energy-Aware Adaptive Sampling Algorithm (EASA) from the Adaptive Sampling Algorithm (ASA) [6]. The optimal sampling rate FN is calculated by Nyquist Theorem. EASA stabilizes the energy levels of sensor nodes at 60% after 36 days of operation, the number of ASA is 20%.

In [7] Taimur Hafeez et el propose a real-time Adaptive Window Based Sampling (AWBS) algorithm to dynamically sample IoT time-series data on the edge devices. Sampling on the edge servers is more intelligent than on the IoT nodes. But the disadvance of this solution is the sampling and anomaly detection is performed by edge server so that it takes some time from the anomaly to the edge devices to recognize and handling it. The result of [7] shows that AWBS effectively reducing the data to 6.91%.

A. *Adaptive Sampling Definition*

Given the data set M, periodic sampling is the triggering of the sensor component's collection process for a fixed period of time T, where T is called the sampling time. This time interval can be 1 second, 1 minute, 1 hour, depending on the system and the settings, the i^{th} data sample is collected at time $t_i = i*T$. For example, T = 2s, sample M_5 will be collected at time $t_5 = 5*2 = 10s$. Because of its simplicity that periodic sampling is widely used.

However, if the context is in the case of battery-based IoTs devices with a limited power source and low processing capacity, it presents many limitations. For example, if

consecutive values (M_{i-1}, M_i, M_{i+1}) are all unchanged, it can take a lot of energy to collect these "nonvalue" data.

TABLE II. TABLE OF NOTATIONS

Notation	Description
M	The input dataset, includes data points (M_1, M_2, M_3,… M_n)
M_i	The i^{th} data point on M which collected at time T_i. This is the last data point of M at the consider time
M'	M' is the reconstructed dataset of M after performing the adaptive sampling
f(M)	The estimation models
T_{i+1}	Sampling time of data collection at time i + 1
ERR	The difference between M' and M

If we take the sampling time T small e.g., 1 second, a large amount of data will be generated and distributed over the network for processing and storage, which can seriously affect the battery of the device. But if the sampling time is large, e.g., 5 second, the change of data cannot be detected and collected.

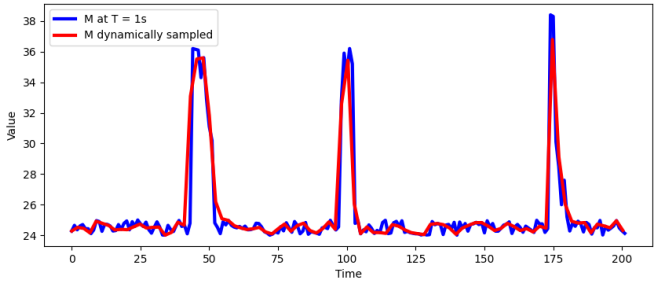

Fig. 3. The adaptive sampling example

The Figure 2 shows the fixed sampling at 1 second and 5 second. At this figure, if the sampling is 5 second the data at time T_{45}, T_{102}, and T_{107} was not sensed and sent although the data at these times is high value. Therefore, the choice of T is difficult, it needs to depend on the change of data over time.

For all these reasons, adaptive sampling was developed to dynamically change the sampling time to reduce the amount of data collected and dramatically save battery life. Figure 3 indicates that data trends have been preserved and that most periodic sampling concerns have been resolved. Adaptive sampling defines a dynamic T_{i+1} sampling time in the interval [T_{min}, T_{max}], calculated from a data stream M based on f(M). Adaptive sampling aims to provide a maximum value of T_{i+1}

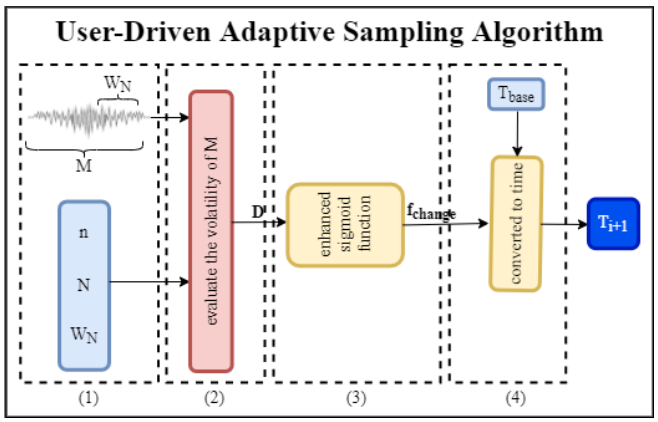

Fig. 4. An overview of UDASA

$\in$ [T_{min}, T_{max}] to collect sample M_{i+1} from M based on f(M) while minimizing ERR to ensure accuracy of f(M). Therefore, the problem is summarized with the following equation [2] [4]:

$$T_{i+1} = \underset{T_{i+1}}{argmax}\{f(M_i, M, ERR) \mid argmin\{ERR\}, \quad T_{i+1} \in [T_{max}, T_{min}]\}$$

The accuracy of data after adaptive sampling is expressed as ERR; if ERR → 0 results in sampling time T_{i+1} → T_{min}, the purpose of adaptive sampling is defeated. As a result, we need an algorithm capable of balancing data accuracy and energy savings for the device.

B. *User-Driven Adaptive Sampling Algorithm*

1) *Related definitions*

Normalized Mean Error (NME) [2] represents the match between the original data and the data after being reconstructed using the adaptive sampling algorithm. It is defined as follows:

$$NME = \frac{1}{n}\sum_{i=1}^{n} |\hat{x}_i - x_i| * 100\%$$

Here, $\hat{x}_i$ denotes the normalized ith data in the reconstructed dataset, x_i represents the normalized ith data in the original dataset, and n is the total data. The lower the NME value, the more accurately the reproducible dataset from the algorithm. Our team uses the python language *scipy.interpolate* library to perform simple interpolation before calculating the NME.

However, due to different numerical ranges of the data, for example, the measured lux is usually between 0 and 20000, the pH is between 0 and 14. Therefore, a normalization step is necessary. before calculating the NME value. The following is how the value of x_i in the data collection M is normalized:

$$x_i = \frac{x_i - \min(M)}{\max(M) - \min(M)}$$

Sampling Fraction (SF) [2] defines the proportion of samples remaining in the data set after using UDASA. It is defined as follows:

$$SF = \frac{\hat{m}}{n}$$

Where $\hat{m}$ is the size of the data after adaptive sampling. The lower SF value, the more data is saved, and the accuracy degrades.

2) *An overview of the algorithm*

In this segment, we research the issues surrounding UDASA to reimplement it on our devices.

First, the algorithm evaluates the dataset's volatility based on the input parameters (1) (Figure 4) by comparing the shift in Median Absolute Deviation between the M_i data and the average of each deviation over the last N data points (2). Through the advanced sigmoid function (3), D will change the f_{change} from 1 → n. n, referred to as the saving level; users can fully alter or modify n depending on the desired level; the larger n, the greater the savings. Finally, a conversion function converts the change value to the sampling time (4). The more the data fluctuates, the closer the sampling time gets to the minimum value of the range (T_{base}, for periodic sampling, the

sampling time is T_{base}). In contrast, if the data fluctuates just slightly, the sampling time will gradually approach the maximum value of the range (n* T_{base}). In summary, $T_{i+1} \in [T_{base}, n*T_{base}]$ with n user input. The algorithm solved the trade-off mentioned above between accuracy and energy savings.

III. Proposed System

A. System design

In this section, we will demonstrate our IoT system namely SmartCities based on Libelium devices. In this system the sensor values e.g., temperature, humidity, light, … are collected, sent, and stored to a cloud database via the gateway. The connection beside sensor nodes and gateway is Zigbee, and gateway connects to cloud by Ethernet connection. The database is put on UIT cloud. After that, we use the UDASA on this system's data. The web application is also used to run UDASA. The aim of deploying UDASA is to experiment and to evaluate the algorithm from theory to practice. In general, the system is made up of three major components: a cloud data center, a gateway, and an end-device. The Figure 5 shows the high-level design of SmartCities system.

Cloud data center: A component used to deploy client-side services such as web services, where users can enter parameters for UDASA, demonstrating versatility by using different parameters for each type of sensor and managing devices connected to the system. The sensors are included, and all of them are saved in the database. In terms of the database, it is the location where device information, sensor values for each device, and other data are stored. Furthermore, the web application allows for real-time analysis of the results obtained by the use of UDASA, as well as visual representation of the device's battery life while using and not using the algorithm.

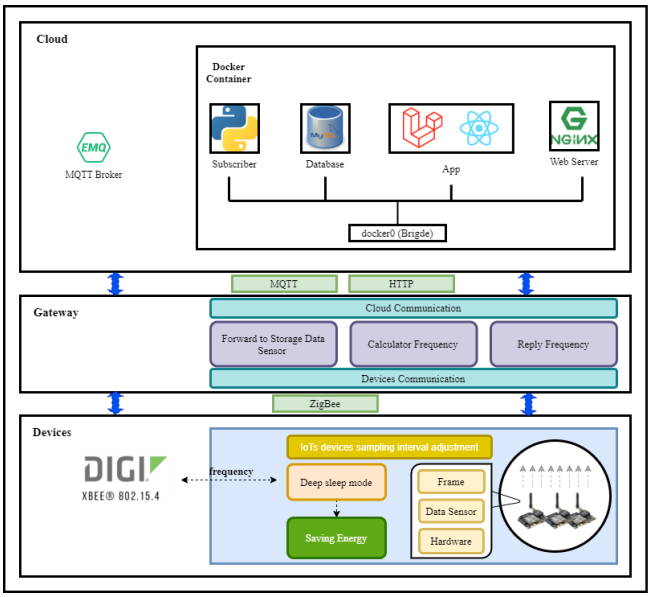

Fig. 5. The high-level design of the proposed system model

Gateway: This component is located between the cloud data center and the end-device. The gateway structure integrates embedded devices to receive sensor values from the end-device portion, forwards these values to the cloud component to update the database, and refreshes the web application. This part also performs the most important task - computation of sampling frequency values from UDASA, which is determined after each data point is obtained to ensure the algorithm is computed with the continuous change of values from the setting. The sampling frequency is continually suggested, and the result is returned to the end-device.

End-device: The end-device is made up of sensors that collect and send values to the gateway, and devices that attach to the gateway in a star pattern. The primary function of this component is to acquire constantly changing sampling frequencies from the gateway component in order to change the device's sampling frequency, well with the ultimate goal of saving energy.

B. System implementation

This section describes the proposed system's use of modern technologies:

Cloud data center: In consideration of the web application, we use a Laravel framework-based backend server. The database is built with MySQL, as recommended by Laravel, and React.js, a JavaScript library for creating user interfaces. MQTT, an ideal communication channel for IoTs applications, will be deployed in this component to communicate between the cloud data center and the gateway. For ease of deployment, all will be deployed on the Docker Container.

Gateway: We use a device with the processor: Intel (R) Core (TM) i5-6200U CPU @ 2.30 GHz, 2401 MHz, 8 GB of RAM attached to Waspmote Gateway 802.15.4 PRO to receive data from the device through the Zigbee protocol.

End-device: Waspmote is a libelium advanced chip used in this component to add sensors by combining additional circuit boards (Smart Cities PRO, Smart Agriculture) (BME, lux). To communicate with the gateway, it uses the Xbee 802.15.4 module.

IV. Experimental Results

In this section, we demonstrate the results obtained through our experiments and data from our system. Part A re-implements UDASA on NOAA datasets to apply it to our specific scenarios. The next one, Part B performs UDASA on our Smart Cities - UiTiOt datasets to further evaluate the obtained parameters of the algorithm. In Part C, we implement UDASA just done above on our system and tested for a sensor. Finally, like Part C, but Part D will test collect a variety of sensors. Both two scenarios are adjusted for different savings levels on our system.

A. Re-implement UDASA on NOAA datasets

"National Oceanic and Atmospheric Administration (NOAA)" provides a set of real-time data about water quality in a place called "Jamstown". Data is maintained from 2008 to present, interval sampling is 1 hour. We extracted the DO and turbidity values from December 15, 2016 to March 15, 2017.

From [2]'s pseudo-code, we used the python language to re-implement UDASA. By different methods, we evaluated to know what is the correct UDASA on the NOAA dataset from the suggestion of [2]. Our best results are shown in Table III.

The energy-saving value is changed at 1, 3, 5, 10, 15, 20, with 1 indicating no savings and this being the initial dataset. The window size is set to 30.

TABLE III. OUR IMPLEMENTATION RESULTS OF UDASA ON NOAA DATASETS

Datasets	Metrics	Saving Levels					
		1	3	5	10	15	20
DO	Number of Samples	2090	1001	588	263	173	134
	SF	1	0.48	0.28	0.13	0.08	0.06
	NME	0	1.73	2.52	4.85	5.54	5.77
Turbinity	Number of Samples	2070	978	552	238	167	133
	SF	1	0.47	0.27	0.11	0.08	0.06
	NME	0	2.46	3.56	6.76	6.92	6.93

B. *Applying UDASA on Smart Cities - UiTiOt datasets*

We continue to use UDASA in section A on the Smart Cities - UiTiOt dataset to assess the potential for balancing saving data points obtained while maintaining the datasets' accuracy.

TABLE IV. OUR IMPLEMENTATION RESULTS OF UDASA ON SMART CITIES - UITIOT DATASETS

Datasets	Metrics	Saving Levels					
		1	3	5	10	15	20
Lux	Number of Samples	5184	2571	1648	740	422	297
	SF	1	0.5	0.32	0.14	0.08	0.06
	NME	0	3.93	4.7	7.47	11.43	14.56
Temperature	Number of Samples	5116	2272	1179	539	370	285
	SF	1	0.44	0.23	0.11	0.07	0.06
	NME	0	1.35	1.89	2.91	3.93	4.88
Humidity	Number of Samples	5116	2200	1126	540	370	285
	SF	1	0.43	0.22	0.11	0.07	0.06
	NME	0	1.2	1.8	3.02	4.18	5.44
Pressure	Number of Samples	5116	2360	1278	542	370	285
	SF	1	0.46	0.25	0.11	0.07	0.06
	NME	0	0.43	0.62	1.08	1.7	2.57

Smart Cities - UiTiOt provides a set of real-time data about different environmental data on the 12th floor of Building E, University of Information Technology - VNU-HCM. The dataset was deployed from July 2020 to the present, with interval sampling every 15 minutes. We will extract the typical values as lux, temperature, humidity, pressure from March 1 to April 23, 2021. The reason for this implementation is that these values would fluctuate significantly during the peak dry season in Ho Chi Minh City. The energy-saving levels are set to 1, 3, 5, 10, 15, 20, and window size is 30, with a T_{base} of 15 minutes applied to the 4 data sets mentioned above. Figure 6 shows that the data precision is maintained.

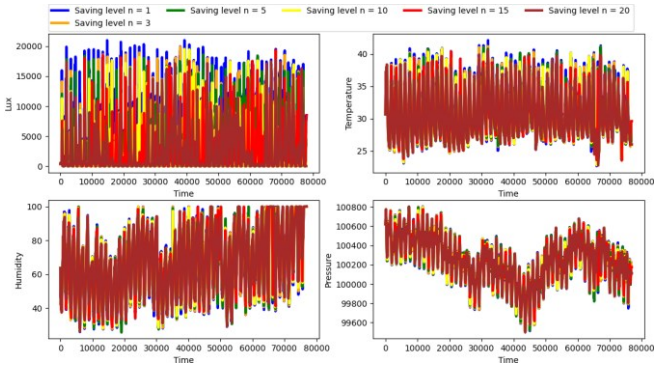

Fig. 6. The similarity among data trends with different saving levels on lux, temperature, humidity, pressure datasets

Changes in the energy-saving level n significantly decreased the number of data points. E.g., at temperature, n is 3, the number of data points to collect is just around 44% of the original data (from 5116 data points to 2272 data points). The NME value is 1.35%, implying that the dataset after using the UDASA is only 1.35% different from the original dataset and has an accuracy of 98.65%. As the n value is increased to 5, the number of data points decreases by approximately 4.3 times, and the precision is marginally reduced to 98.11%. Similarly, as n is increased in steps until it reaches 20, the accuracy drops to 95.12%.

The sensor values for lux, humidity, and pressure were similar, particularly pressure value when n = 3, NME value reached 0.43%, equivalent to 99.57% accuracy. Figure 6 depicts the trend of the datasets, where the data at lux fluctuates significantly (from more than 20,000 to 0), so the NME value of lux is higher than the other values when we surveyed (14.56% with n = 20). See Table V for more information.

C. *UDASA is deployed on the system in case to collect a sensor value*

TABLE V. LUX DATASET WITH DIFFERENT SAVING LEVELS

Datasets	Metrics	Saving Levels			
		1	3	5	10
Lux	Number of Samples	384	175	116	64
	SF	1	0.56	0.35	0.1
	NME	0	3.83	13.9	28.11
	Decline of pin	0	5	6	6

We continue to use UDASA on our systems with devices from the libelium suite to measure the device's actual power consumption. In this case, we use waspmote to collect the lux values. Two waspmotes were gathered at the same time and location; one waspmote will be collected normally, while the other will be deployed on UDASA. Our system will alternately set the value of energy-saving level n to 1, 3, 5, and 10. The initial dataset for the algorithm is from the Smart

Cities - UiTiOt datasets, November 1, 2020, from 0:00 to 8:30.

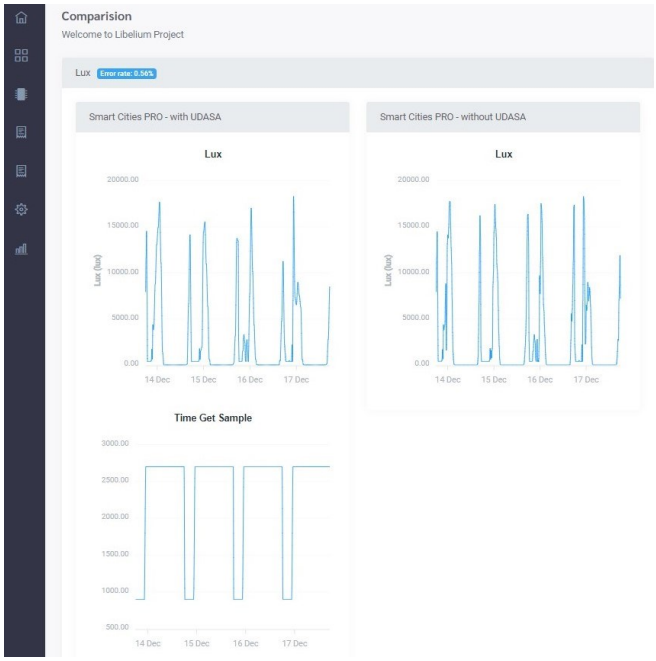

Fig. 7. Data trends are shown on our system

Our web application collects and displays all scenariors's results (see Figure 7). Table V displays the reports from each case over the course of 4 days (total of 12 days). For savings levels of 3, 5, and 10, the NME values are 3.83%, 13.9%, and 28.11%, respectively. Specifically, the highest battery saving that the device will view is 6% over 4 days.

D. UDASA is deployed on the system in case to collect many sensor values

In fact, the waspmote collects not just one sensor value but many. We conducted to collect three values of temperature, humidity, and pressure simultaneously. After each collection, we will select the lowest sampling frequency value to adjust the sampling frequency on the waspmote. Since the T_{base} value of the UDASA output is the lowest, the accuracy of the three datasets is still assured after using UDASA.

TABLE VI. TEMPERATURE, HUMIDITY, PRESSURE DATASETS WITH DIFFERENT SAVING LEVELS

Datasets	**Metrics**	**Saving Levels**			
		1	**3**	**5**	**10**
	Number of Samples	385	217	136	40
	SF	1	0.56	0.35	0.1
Temperature	NME	0	5.66	7.91	15.44
Humidity			7.84	10.47	15.34
Pressure			2.07	7.66	16.89
	Decline of pin	0	7	6	9

Table VI show that n = 3, the minimum NME rate of temperature was 5.66% (accuracy is 94.34%), and it only needed to collect 56% of the initial amount of data. When n = 10, the sum of data collected is only 10%, but NME is 15.44% (accuracy is 84.56%). The precision was severely reduced. Same for humidity and pressure values. In general, the error rate is higher than Part C due to the need to balance the 3 sensor values when calculating the sampling frequency. Since the system requires more power for the sensors component, the maximum energy savings achieved by the display device over 4 days is 9%.

V. CONCLUSIONS

In this work, we applied the UDASA [2] into the environmental monitoring station namely Smart Cities – UiTiOt system. UDASA is the User-Driven Adaptive Sampling Algorithm that estimates in realtime the optimal sampling frequency for IoT devices based on the changes of data in history. This algorithm strikes a balance between the size of the data collected and the corresponding energy savings. From the pseudo-code of UDASA, we re-implement this algorithm by python programming language, and apply it into our system. We implement several scenarios with different types of sensors. In the only one Lux sensor case, the system only collected 64 data points when if without UDASA, the system must collect 384 data points. And the energy-saving at this scenario is 6% corresponding to n = 10. In the scenario which uses 3 sensors Humidity, Temperature, and Pressure, our best result is 9% energy-saving when n = 10, and the sum of data collected is only 10% compared to fixed time sampling.

VI. REFERENCES

[1] "Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025," [Online]. Available: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/. [Accessed 25 April 2021].

[2] L. Kim-Hung and Q. Le-Trung, "User-Driven Adaptive Sampling for Massive Internet of Things," *IEEE Access,* vol. 8, pp. 135798-135810, 2020.

[3] D. Giouroukis, A. Dadiani, J. Traub, S. Zeuch and V. Markl, "A survey of adaptive sampling and filtering algorithms for the internet of things," *Proceedings of the 14th ACM International Conference on Distributed and Event-based Systems,* p. 27–38, July 2020.

[4] D. Trihinas, G. Pallis and M. D. Dikaiakos, "Low-cost adaptive monitoring techniques for the Internet of Things," *IEEE Transactions on Services Computing,* vol. 14, pp. 487-501, 2021.

[5] B. Srbinovski, M. Magno, F. Edwards-Murphy, V. Pakrashi and E. Popovici, "An energy aware adaptive sampling algorithm for energy harvesting wsn with energy hungry sensors," *Sensors,* vol. 16, p. 448, 2016.

[6] C. Alippi, G. Anastasi, C. Galperti, F. Mancini and M. Roveri, "Adaptive sampling for energy conservation in wireless sensor networks for snow monitoring applications," *MASS,* pp. 1-6, 2007.

[7] T. Hafeez, G. McArdle and L. Xu, "Adaptive window based sampling on the edge for Internet of Things data streams," in *11th International Conference on Network of the Future (NoF)*, 2020, pp. 105-109.