UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Afsana Khan

Synthetic Sensor Data Generation for Authentic IoT Device Emulation

Master's Thesis (30 ECTS)

Supervisor(s): Pelle Jakovits, PhD
Mainak Adhikari, PhD

Tartu 2021

Synthetic Sensor Data Generation for Authentic IoT Device Emulation

Abstract: The Internet of Things (IoT) is the concept of connecting everyday physical objects to the internet and making them capable of identifying and communicating with other devices. The rise of the Internet of Things in the past few decades is commendable. This significant growth has resulted in IoT becoming a budding industry that provides a variety of products, services, and systems and therefore requires quality assurance. These systems are very complex in nature and are used in real-world environments. For these reasons, IoT systems and devices are obliged to testing. The implicit heterogeneity of IoT systems, which typically consist of many thousands of interacting actuators, sensors, and people, makes this an incredibly challenging task. A very important aspect when testing Internet of Things systems at the system level is to include the behavior exhibited by local entities such as IoT devices. However, physically deploying such a huge number of IoT devices is not always feasible in terms of cost, resource availability, and time. To solve this particular problem, it is essential to make use of data generators for synthetically generating IoT data that can emulate the behavior of real IoT devices. Motivated by this challenge, this thesis focuses on evaluating open source tools for generating real-like IoT data based on existing captured data traces, weighing their pros and cons, and finally proposing an approach that would perform more efficiently to achieve the required goal as well as be reusable. In addition, the proposed approach is evaluated by generating synthetic data for some existing datasets and the results are compared.

Keywords: Internet of Things, Synthetic Data Generation, IoT System Testing, IoT Data Emulation

CERCS: P170 Computer science, numerical analysis, systems, control

Sünteetiliste sensoriandmete genereerimine IoT-seadmete autentseks emuleerimiseks

Lühikokkuvõte: Asjade internet (IoT) on igapäevaelu füüsiliste objektide ühendamine üle interneti, et võimaldada neil üksteist identifitseerida ning omavahel suhelda. Asjade interneti areng viimastel aastakümnetel on kiiduväärt. Selle märkimisväärne kasv on toonud kaasa IoT muutumise kasvavaks tööstusks, luues võimalused uute toodete, teenuste ja süsteemise loomise ning nõuab seetõttu kvaliteedi tagamist. Need süsteemid on loomult väga keerulised ning neid kasutatakse juba reaalsestes keskkondades. Nendel põhjustel on tähtis IoT-süsteeme ja seadmeid testida. IoT-süsteemide heterogeensus, mis võivad koosneda paljudest tuhandetest interaktiivstest täituritest, anduritest ja kasutajatest, teeb selle ülesande uskumatult keeruliseks. Väga oluline aspekt IoT süsteemide testimisel on kohalike seadmete kaasamine testimisel, et arvesse võtta seadmete individuaalne käitumine. Sellise suure hulga IoT seadmete füüsiline kasutuselevõtt ei ole aga siiski alati teostatav kulude, ressursside kättesaadavuse ja ajakulu tõttu. Selle konkreetse probleemi lahendamiseks on oluline kasutada andmete generaatoreid, mis võimaldaksid genereerida sünteetilisi IoT andmeid, mis omakoda võimaldaks emuleerida reaalsete IoT seadmete käitumist. Sellest väljakutsest motiveeritult keskendub see lõputöö avatud lähtekoodiga tööriistade hindamisele päris-andmete sarnaste andmete genereerimiseks, mis põhinevad olemasolevate IoT andmete jälgimisel, võrreldakse nende lahenduste plusse ja miinuseid ning lisaks pakutakse välja lähenemine, mis toimiks tõhusamalt vajalike nõuete täitmiseks ja oleks korduvkasutatav. Lisaks valideeritakse välja pakutud lahendus genereerides andmed olemasolevate andmekogude põhjal ning võrrelddakse tulemusi.

Võtmesõnad: Asjade Internet, andmete sünteetiline genereerimine, Asjade Interneti süsteemide testimine, Asjade Interneti seadmete emuleerimine

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Contents

1 Introduction **6**

1.1 Motivation 7

1.2 Contribution to Thesis 9

1.3 Organization of Thesis 9

2 Background **11**

2.1 Internet of Things 11

2.2 IoT data sources 13

2.3 Synthetic data generation 15

3 Experimental Analysis of Existing Data Generation Tools **19**

3.1 IoT Data Simulator 20

3.2 CTGAN 22

3.3 Synsys 25

3.4 log-synth 27

3.5 Iosynth 29

3.6 Conclusion from the Experiments 32

4 Proposed Approach for Synthetic IoT Data Generation **34**

4.1 Architectural Workflow of the Proposed Approach 35
4.1.1 Analysis of Dataset and Creation of Device Profiles 38
4.1.2 Execution and Scheduling of the Orchestrator 39
4.2 Modifications Made in the Data Generator 41
4.3 Containerized Deployment 42

5 Evaluation of the Proposed Approach **44**
5.1 Delta Centre Electricity Usage Dataset 44
5.2 Delta Centre Solar Level Dataset 50

6 Conclusion and Future Work **56**

References **59**

Appendices **60**
Repository for Source Codes 60
List of Tables 61
List of Figures 62
Licence 64

1 Introduction

The IoT is the network of physical devices, vehicles, home appliances, and other items embedded with electronics, software, sensors, actuators, and connectivity which enables these things to connect and exchange data. The term "Thing" in "Internet of Things" is used quite broadly. For example, a thing within the IoT could be a person with a heart monitor implant, a vehicle that has built-in sensors to alert the driver when tire pressure is low — or any other natural or man-made object that an IP address can be assigned to, thus gaining the ability to transfer data over a network. As a result, it is becoming increasingly easy to create opportunities to directly integrate the physical world into computer-based systems which results in improvements, efficiency, economic benefits, and reduced human exertion. The IoT has gained momentum in recent years due to internet availability and an increase in the number of devices generating a huge amount of data to be transferred. IoT devices are growing at 30 percent year on year and will reach US$3010 billion[1]. People, government, and businesses will be hugely affected by IoT in the coming years resulting in smart cities, smart healthcare, intelligent energy management and smart grids, automotive traffic management, and so on.

The arrival of the Internet of Things has resulted in the expansion of IoT infrastructures. Raw data is being collected at an increasing rate from a multitude of devices and sensors – sources that include connected homes, smart meters, manufacturing, healthcare, mobile devices, vehicles, and more. The ingestion and integration of raw data, the extraction of valuable information from the raw data, and planning for infrastructure capacity and analytic capability for analyzing this data are new challenges in the era of big data. The development of infrastructure support and analytical tools to handle IoT data, which are naturally large and complex, has become essential.

IoT devices have been producing an enormous amount of data continuously. These data need to be saved and analyzed for making various decisions in the future. To achieve this goal, IoT systems have been upsurging. The complexity of these IoT systems is increasing exponentially as they grow[2]. To test and evaluate the performance of the IoT systems, it is necessary to simulate real-world workload models, which can be created

using historic data, types of devices, usage patterns, geographic spread, and so on. One of the main challenges of testing and evaluating IoT infrastructures is the availability of real-life IoT traces. Although IoT devices generate a massive amount of data, it is not always convenient to get access to those. The requirement of using large amounts of IoT traces leads to the set up of numerous physical devices, which is not always feasible in terms of cost. The possible solution to these problems is to generate synthetic IoT data that have characteristics that are similar to the original data but have values that are not obtained by direct measurement. However, IoT data are not always random in nature. Due to the heterogeneity of IoT data sources, it is often observed that IoT data, rather than producing random data, follows a specific pattern or exhibits spikes and falls over specific time periods. For instance, if the data from the sensors which generate street noise levels are collected over the time period of a day, it can be observed that the noise levels are much higher during the peak hours when traffic congestion on streets is more, while the noise levels decrease during the early morning or late night hours.

1.1 Motivation

The motivation behind this thesis comes from the CIISIM project, which is an Estonian Research Council-funded research project under the RITA program. The project's objective is to expand the Cyber Range extension for conducting cyber defense exercises in order to strengthen Critical Infrastructure Providers' capabilities to protect society from potential threats posed by the use of new technologies such as the Internet of Things and next-generation mobile connectivity technologies. The project involves the creation of a Physical Village which consists of several Critical Infrastructure applications that utilize a variety of end-device connectivity technologies. Additionally, virtual replicas of the Physical Village are created by virtualization of both hardware components and communication links which enables training at a lower cost. Since the hardware components are virtualized, this means these components need synthetic data that behaves like actual IoT data to make scenarios and trainings look more realistic. As a result, a suitable synthetic data generation approach is required in order to emulate the real IoT

devices of the physical village. The data generation approach has to meet the following requirements:

- Generate IoT data taking into account the observable seasonality and other patterns.
- Have the ability to configure properties such as the type of data that must be produced, the distribution that the generated data must adopt, and the frequency with which the data must be generated.
- Have the functionality to publish generated data to MQTT since MQTT is one of the standard IoT protocols for sending and receiving data and also since MQTT protocol was decided to be used in the project.
- Can be deployed in a containerized manner (e.g. using Docker) to make the solution easy to install, deploy and scale.
- Be able to automate the deployment of emulated devices.

To emulate IoT systems as closely as possible, it is important for the synthetic data produced to share similar characteristics. Although a considerable amount of research has been previously done on synthetic data generation in general and several data generators are being used for testing, these data generators are either too general for a specific use or too specialized for one type of problem and thus hardly reusable. The goal of this thesis is to fill this gap, investigate and evaluate existing data generators as well as design a suitable approach for generating synthetic IoT data that fulfills the requirements of the CIISIM project. After which, the goal is to implement the approach and evaluate the results.
The following research questions need to be addressed while designing an effective approach to generate synthetic data:

1. How to generate synthetic IoT data with adjustable frequency without the need to set up devices physically?

2. How to generate synthetic data to mimic the natural data generated by IoT devices, taking into account the observable seasonality and other patterns?

1.2 Contribution to Thesis

This thesis addresses the research challenges mentioned above and aims to bring forward a solution for creating synthetic IoT data that would mimic the behavior of real data generated by IoT devices. The solution is expected to process existing data traces and generate similar behaving IoT data streams with high volume and frequency as well as support customizing the generated data stream, including volume & frequency, structure, and randomizing certain fields of records. The main contribution of this work can be summarized as follows:

- A comprehensive evaluation of available open-source tools has been done for generating real-like IoT data based on existing captured data traces and played back to simulate real data in an IoT network.
- Considering the pros and cons of the existing data generators, a solution has been proposed which would satisfy the requirements for synthetic IoT data generation.
- The proposed solution is tested on chosen use cases and similar behaving relevant data is generated.
- The generated synthetic data is evaluated to see how closely it resembles the real traces in terms of behavior.

1.3 Organization of Thesis

The following chapter provides an overview of the Internet of Things, the motivation behind this thesis, research challenges and contributions to the thesis. A brief introduction of IoT systems, possible IoT data sources, and state-of-the-art approaches to synthetic

data generation is given in Chapter 2, as well as an overview of existing data generators is provided. Chapter 3 covers an experimental analysis of available data generators by creating synthetic data using a particular IoT dataset and weighing their benefits and drawbacks. Later in chapter 4, a solution for synthetic IoT data generation is proposed and illustrated. Chapter 5 includes an evaluation of the proposed approach by using it on existing datasets and analyzing the results to determine its behavioral resemblance to real IoT traces. Finally, chapter 6 concludes the overall work by highlighting the accomplishments of the thesis and discussing potential applications and future extensions.

2 Background

This chapter provides a background on the concept of the Internet of Things (IoT) and the IoT systems, sources of IoT data, and related works on synthetic IoT data generation and existing data generators. The first section discusses IoT in general and mentions the examples of some IoT systems. In the second section, possible sources from where IoT data could be obtained are covered. The last section explores the methodologies for the generation of synthetic data and gives an overview of the current data generators that have been developed and are in use.

2.1 Internet of Things

The Internet of Things (IoT) is a distributed ICT (Information and Communication Technology) system that integrates sensors, computing devices, algorithms, and physical objects known as the $Things$ which are uniquely identifiable. The $Things$ have the ability to collect and transfer data over connected systems without any human intervention, thus offering autonomous data processing abilities. A communication network is one of the key elements of an IoT system that allows information flow among a large number of sensors, actuators, devices controllers, and data storage. Many IoT devices require mostly one-way data transferability, whereas some applications operating in sensor-actuator modes require bi-directional data transfer. IoT systems could be deployed to support numerous applications; ranging from simple home automation tasks to life-saving tasks where implanted sensors within a human body could be used to monitor critical human organs. In today's world IoT systems have been introduced in many domains including social, civic, health, education, industrial, transport, defense, smart agriculture, and smart cities.

Tele-health is a major use of IoT applications where a range of medical devices both implanted and on-body devices could be used to gather statistics and control different

medical purposes. Cloud computing systems coupled with medical wearable devices and home/workplace networks not only monitor the health of a person but also offer timely advice in real-time.

Smart grid is an example of IoT system in the utility areas which is the electricity grid. The use of effective condition monitoring systems in the smart grid has attracted other utility sectors to think about deploying such facilities to improve service reliability and reduce operational costs. IoT systems are also being used in the utility areas such as water and gas distribution services, oil pipeline monitoring, and road traffic management.

Agriculture and livestock monitoring are two major areas where field-deployable IoT systems can make significant contributions to improve agricultural output, reduce crop failures, better farm animal's health and safety which can benefit the industry significantly.

Smarty city is a major and complex IoT application domain where many civic facilities and service management functionalities need to be integrated under a single distributed IoT/ICT platform [3]. Smart city IoT platforms are designed to support enhanced civic facilities to promote quality of life, safety, and security in order to deliver a sustainable and livable city for citizens. A smart city provides many services where IoT solutions need to be applied effectively to offer these services in a cost-effective and timely manner. Some of the key services are smart and sustainable energy, energy efficiency intelligent traffic, transportation, and parking management, environmental monitoring, and other civic facilities.

The project CISIIM also deals with the aspect of a Smart city. It includes creating a Physical village which would be a prototype of a smart city having the functions of monitoring road-noise, solar levels, water levels, vehicular movement, and so on. The physical village will also have virtualized copies as mentioned earlier.

2.2 IoT data sources

The ever-increasing heterogeneous IoT devices are now leading to the generation of a huge amount of IoT data. Eventually, these data are being collected and stored in various formats for further analysis and processing which might help in decision-making purposes. Besides, existing IoT data can also be used for performance testing of IoT systems. One of the major sources of IoT data is the real IoT systems which have physically set up IoT devices continuously generating data and communicating among themselves. These data are stored in storage systems and datasets are created. One example of such a dataset can be the Intel Lab Data [4] which was created using the data generated from 54 sensors deployed in the Intel Berkeley Research lab between February 28th and April 5th, 2004. Mica2Dot sensors with weatherboards collected timestamped topology information, along with humidity, temperature, light, and voltage values once every 31 seconds. Data was collected using the TinyDB in-network query processing system, built on the TinyOS platform. Another example could be the "IoT analytics - assessing IoT device security and privacy posture" dataset [5] created by the UNSW Sydney researchers by performing performed extensive research into the network behavior of IoT devices and created a set of IoT device traces and profiles datasets. They include captured packet logs, attack traces, and also a set of profiles for commonly used consumer-grade IoT devices, which may be very useful for emulating the network traffic of IoT devices. Similarly, the IoT Network Dataset includes object information (such as owner or category), the traces on the positions and the timestamps of the devices, the list of all the applications envisioned in a Smart City scenario, the object's profiles (expressed as a set of available services and possible applications requests) and an adjacent matrix of relationships for the Social IoT. The total number of objects comprehends a total of 16216 devices, of which 14600 are from private users and 1616 are from public services.

A possible source for IoT data can be testbeds. A testbed is a platform for conducting rigorous, transparent, and replicable testing of scientific theories, computational tools, and new technologies. An IoT testbed is a subsystem of a real IoT system that physical IoT

devices set up in a limited amount but can be used as a prototype of a real system. Such testbeds can be used for testing the performance of real IoT systems. The data generated from such testbeds can also be collected and stored for future use. One example of such a testbed is the "SmartSantander -Large-scale Smart City deployment" testbed [6] located in Santander, a seaside town settled in the north of Spain. With a population of nearly 200,000 inhabitants, this city was chosen to deploy an experimental test facility (i.e. open laboratory) for the research and experimentation of big-scale architectures, in the context of a Smart City environment. The data collected from this testbed contain information on environmental monitoring, traffic monitoring, garbage management, electromagnetic exposure, indoor and outdoor parking, presence and luminosity, etc. The "NITOS - Heterogeneous LoRa and Wireless Sensor Network" testbed is a set of heterogeneous resources including nodes equipped with Wi-Fi, WiMAX, LTE and Bluetooth, 4G/5G terminals, software-defined radios, SDN resources, cloud infrastructure, and two wireless sensor networks comprised by commercial and custom made sensor platforms. On top of these diverse and heterogeneous resources, NITOS brings a vast number of experimental datasets generated through the various experiments conducted all these years that NITOS has been operating. The sensors used in this testbed provide readings of luminosity levels, ambient temperature, relative humidity, conductivity, atmospheric pressure, and so on.

However, during the study of these datasets and testbeds, it was found that most of the data repositories and testbed data were easily accessible. Under such conditions, a reliable source of obtaining IoT data could be creating synthetic data. There have been several studies on the approaches of synthetic data generation and also using data generators to generate realistic synthetic data. Section 2.3 elaborates on this aspect and provides an overview of existing synthetic data generation approaches and data generators.

2.3 Synthetic data generation

Myung et al. [7] proposed a smart home analysis method by utilizing time dependency, usage continuity of daily appliances, and correlation between them to model behavior patterns of service users and generate reliable synthetic data based on their real log data. In this work, data analysis and pre-processing has been done on the Reference Energy Disaggregation Data Set (REDD) [8], which contains smart home usage log from multiple daily appliances for a long time collected from 6 houses; one sensor for each service installed in every house. From the time duration and readings of the sensors for each service, the usage pattern for those services was determined and later synthetic data was generated using the Markov chain algorithm. However, the data analysis method in this approach is complex and as the size of data to be generated increases, too much time is required for pre-processing input data and generating synthetic data which is mentioned by the authors. This could be a disadvantage in cases where it is necessary to emulate real-world IoT devices continuously and for a longer duration of time.

A tool was developed in [9] which was capable of simulating human activities in a smart home environment. By publishing the simulation data through MQTT into the OpenHAB framework, the end researchers can obtain a database of real-looking data. The work consisted of using a simulator that had a graphical user interface where the user could specify the events and actions of the IoT devices, the data from the simulator is then sent over to an MQTT broker and finally to the OpenHAB framework to generate a dataset similar to a real dataset. However, the paper is short on details about how the simulator is used, and it's unclear whether it always requires users to specify IoT device events. The simulator does not create synthetic IoT data automatically, which could be helpful if the goal is to emulate IoT devices.

Haris et. al [2] proposed a simulation environment named Sensyml that is capable of generating big data from cyber-physical models and real-world data. Sensyml is capable of hosting multiple types of sensors with physical and geographical characteris-

tics similar to real sensors, where sensors can be derived from mathematical models or extracted from existing online services. The simulation environment can be integrated into existing hardware infrastructure to test basic functions of the IoT and how the IoT scales up. This work focuses on specific forms of sensor data, such as random numeric or binary data, rather than actual time-series sensor data. The overall work is more concerned with the generation of load in order to test IoT infrastructures rather than actually trying to mimic real IoT devices. W. Anderson et al. [10] designed a synthetic IoT data generation framework that is capable of generating terabytes of structurally similar synthetic data from a highly complex and nested original data source. The IoT synthetic data generation in this work proceeds in two major phases. The first phase is to perform structure and value extraction from the original XML, resulting in a data characterization called the synthesis set. The second phase uses the synthesis set as input to generate and output synthetic XML data. The work in this paper enables the creation of large volume synthetic datasets that retain important characteristics of the data but it does not mention what kind of characteristics are maintained here.

For testing purposes, several open-source data generations and simulation tools are being used to replicate the behavior of actual data traces. The Quality Testing (QT) tool [11] is a software developed for load injection and testing of Data-Intensive Applications (DIAs). It is capable of capturing incoming JSON stream data, generating a model for replaying similarly behaving but random data, and injecting the data back into a stream data processing pipeline. The evaluation of the QT tool is done using the Twitter dataset which consists of streams of tweets and retweets along with their time stamps. The QT tool uses the dataset as input and applies Hidden Markovian Model(HMM) to generate tweets and retweets with new timestamps. The amount of tweets to be generated could also be controlled by the user. Here the contents of the tweets are not changed rather just timestamps are modified in order to perform load testing. The QT tool does not seem like a good fit in case IoT data is n generated in real time while also maintaining the behavioural characteristics. For load testing using it would be fine but in case of emulation this would not be so useful. Han et al. in [12] brought forward an IoT simulation toolkit $DPWSim$, which uses device profiles to emulate the behavior of IoT

devices. $DPWSim$ allows developers to prototype, develop, and test IoT applications using the DPWS technology without the presence of physical devices. A Smart City & IoT Wireless Sensor Network Simulator $CupCarbon$ has been described in [13] which has an objective to simulate wireless IoT and smart city networks and to support creating different environmental scenarios such as fires, gas, accidents. It is useful for designing, debugging, and validating different monitoring, environmental data collection, routing algorithms, and visualizing the results. However both $DPWSim$ and $CupCarbon$ are more like simulators which can be used to protype IoT environments and simulate them but does not necessarily focus on the aspect of observable seasonality and other patterns of IoT devices. The IoT-data-simulator [14], is a tool that consists of interesting features like replaying, existing datasets with modified data (such as updated timestamps, generated ids, etc.), generating datasets of any complexity, and customizing frequency with which data will be sent - based on dataset timestamp properties or just constant time interval. Since the IoT- data simulator can replay datasets with modified timestamps, it can be used for emulation purpose but there is no clear indication if it exhibits the actual patterns of real IoT devices and whether it can be used to emulate multiple devices at the same time. CTGAN [15] is a generative adversarial network-based data synthesizer that can generate synthetic tabular data with high fidelity. CTGAN uses a variational Gaussian Mixture model to detect modes of continuous columns and fully-connected networks, which makes it more efficient. CTGAN could be useful in generating synthetic datasets but the use of it for emulation purpose is quite skeptical. The tool $SynSys$ [16] generates synthetic sensor event data using a real smart home data file as a basis. SynSys uses Hidden Markov Models (HMMs) to generate activity and sensor event sequences and regression to generate timestamps. However, the available repository for using Synsys is too old and requires certain modifications in order to use it. The data generator Log-synth [17] allows generating data based on a schema. Schema-driven generation supports the generation of addresses, dates, foreign key references, unique id numbers, random integers and sort of realistic personal names. The frequency of data generation is customizable and can be used to emulate IoT devices but needs to be experimented first. Another open-source tool $IoSynth$ [18] is IoT device/sensor simulator and synthetic data generator, which also allows generating data based on a schema. Schema-driven

generation supports the generation of addresses, dates, foreign key references, unique id numbers, random integers, sensor data, etc. Synsys, Log-synth, and Iosynth might be used to emulate IoT devices by generating similar behaving synthetic IoT data in real-time but it is likely that some modifications still have to be made in order to use them for further purposes.

The literature review conducted identified a lack of data generation methods that focus on imitating actual behavioral characteristics of IoT devices while also generating IoT data in real-time for emulation. The purpose of this thesis is to investigate which data generators could be actually useful for generating synthetic IoT data, specifically sensor data, to emulate actual IoT devices and even customizing it by combining any of the data generations approaches to adapt it with the requirements of the CISIIM project.

3 Experimental Analysis of Existing Data Generation Tools

In the previous chapter, we looked at some of the current synthetic data generation approaches and methods. Later we chose some open source tools for experimentation. The goal of the experimentation is to analyze existing data generators and select a suitable candidate that can be used for the emulation of IoT devices in the CISIIM project. Also, the analysis and experiments of the existing data generators would help in deciding if a specific data generator could be used for the project directly or if it needs to be reused and adapted to the requirements. The tools were chosen based on how well they met the criteria for generating synthetic IoT data and the availability of source codes. This chapter discusses the experimental analysis of various synthetic data generation tools. Initially, the existing solutions were selected depending on their availability. Only the solutions which were easily accessible and did not require paid service were chosen. From the description of existing data generators described in section 2.3, it was decided to experiment with IoT-data-simulator, CTGAN, Synsys, log-synth, and Iosynth for possessing either of the functionalities like real-time synthetic data generation, configurable dataset input in the form of CSV or text or schema, deployable in docker, accessible only from the command line instead of a GUI and sending generated data to target systems like MQTT. The other tools listed in section 2.3 like DPWSim and CupCarbon were more like simulators which required to be used only via their graphical user interface made them unsuitable to be used for IoT device emulation and also they weren't open source tools. The Quality Testing (QT) tool did not generate synthetic data in real-time and its documentation provided illustration with only the Twitter dataset. The implementation was done using MATLAB and it was quite complex to use the tool with any other dataset and generate results.

The documentation of these solutions was studied and later evaluated by generating synthetic data similar to an open dataset. For performing the experiment, the University of Tartu Delta Centre Electricity Consumption Dataset has been used. It

contains information on the electricity usage level (in Kilowatts) throughout different days from $2020/09/23$ to $2020/11/23$ collected from the sensors installed at the Delta Centre. The Electricity Consumption in the building varies during the hours of the day. The data has different patterns on working days and weekends. However, the aim of the experiment is to try to generate similar electricity consumption data during a particular working day and evaluating if the behavior of the data during the hours resembles the original data. The experimental results are as follows:

3.1 IoT Data Simulator

The IoT-data-simulator [14] was developed to facilitate the simulation of data collected from IoT devices. By replaying existing datasets, it is capable of reproducing identical types of data with changed properties such as timestamps, UUIDs, and sensor readings. In addition, the tool supports relative timestamp properties that are dependent on other timestamp or date properties. It means that data can be replayed with the same time intervals as in the original dataset. To configure and begin generating data with this method, one of two choices is to include a dataset in the form of a JSON or CSV file or a schema describing the structure of the data. If a dataset is provided, the tool replays its contents during the session's execution. If a schema is given, on the other hand, data is generated according to the schema.

To experiment with the IoT data simulator, the University of Tartu Delta Centre Electricity Consumption Dataset has been used as an input in order to generate similar behaving synthetic data.

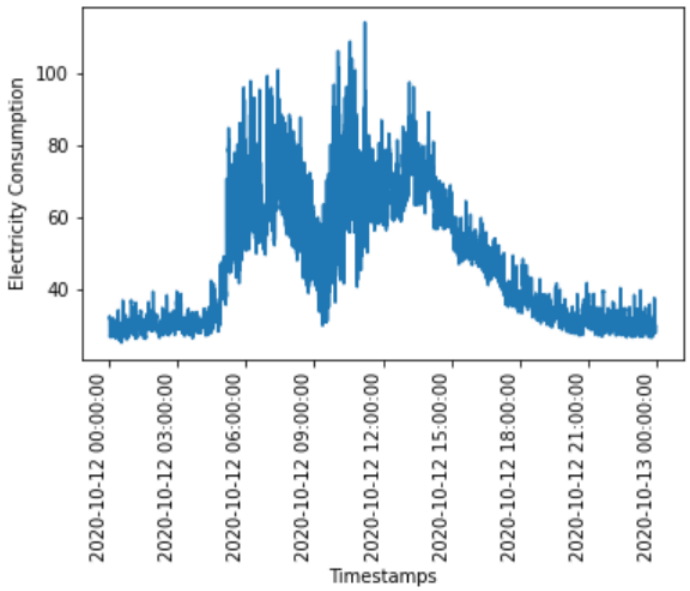


Figure 1. Plot of Electricity Usage Level Data During a Working Day

The figure 13illustrates the pattern of the Electricity Consumption in Delta Centre during the hours in a working day. The IoT data simulator takes this dataset as input and constructs a schema from it, as illustrated in Figure 2where $column_1$ is the timestamps $column_2$ is the electricity consumption(KW) . By modifying the values, the simulator can generate data (figure 3) within the range of the actual dataset. Additionally, the simulator supports sending generated data as a payload to target systems such as AMQP, Kafka, Local object storage, MQTT, and REST

One of the most significant drawbacks of using this simulator is that it can only be used by its graphical user interface. Although the graphical user interface is straightforward and easy to use, it does not fulfill the criteria of the project CIISIM, which required that the simulator be used via the command line or terminal. Additionally, the simulator can generate random values similar to those in the original dataset, but the objective is to generate similar behavior, which is defined as demonstrating similar patterns in behavior among these data. However, no noticeable similarity was observed

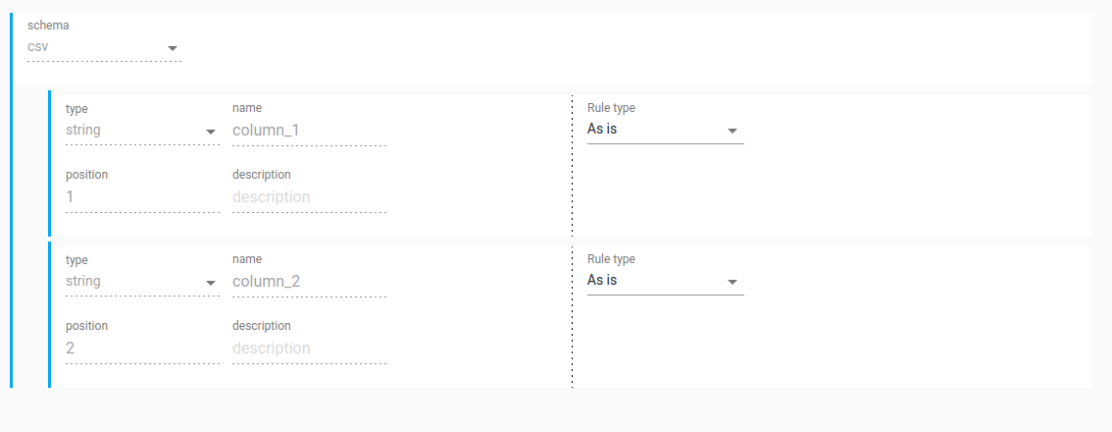


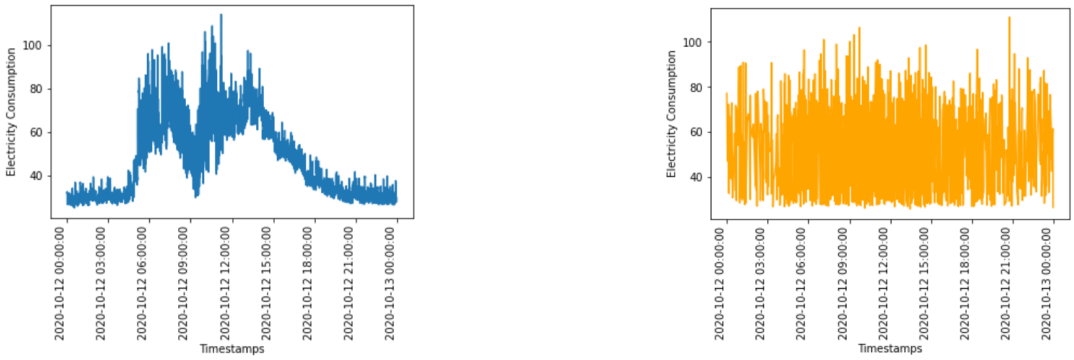
Figure 2. Schema Fetched from the Input Dataset

when comparing the original and synthetic data since the output did not show similar patterns of Electricity consumption during the hours of the day.

3.2 CTGAN

CTGAN is an open-source project from the *Data to AI LAB* at Massachusetts Institute of Technology which is a collection of Deep Learning-based Synthetic Data Generators for single table data, which can learn from real data and generate synthetic clones with high fidelity. CTGAN [15] is a generative adversarial network-based data synthesizer capable of producing high-fidelity synthetic tabular data. CTGAN detects modes of continuous columns and fully-connected networks more efficiently by using a variational Gaussian Mixture model. CTGAN generates synthetic data as output which resembles the data fed to it as input. It expects the input to be a table given as either *numpy.ndarray* or *pandas.dataframe*. The columns in the input can be of only two types:

- **Continuous Columns:** Columns that contain numerical values and which can take any value.
- **Discrete columns:** Columns that only contain a finite number of possible values,



(a) Original Data

(b) Synthetic Data

Figure 3. Comparison of Original and Synthetic Electricity Consumption Data Generated by IoT data simulator

whether these are string values or not.

For trying out the CTGAN module, the University of Tartu Electricity Consumption dataset has been used. Different electricity level usage can be observed at different hours within a day. CTGAN was experimented with to check if it is able to create synthetic data which follows a similar pattern as in the original dataset. The first issue which was found with CTGAN was that it did not work with larger datasets such as the one mentioned here. The CTGAN module was unable to execute and showed a memory error. As a result, to continue with the experimentation, the dataset was shortened and contained electricity usage levels within 8 hours of a particular day. The figure shows the original electricity usage within 8 hours of a single day.

The dataset was then provided as input to the CTGAN module. The timestamp and electricity usage values were the discrete columns here which were given as parameters and 10 epochs were used while fitting the model in CTGAN. Same number of samples were generated as in the original data. The figure compares the synthetically generated electricity usage levels with the original one.

```
from ctgan import CTGANSynthesizer
ctgan = CTGANSynthesizer()
ctgan.fit(data, data.columns, epochs=10)
```

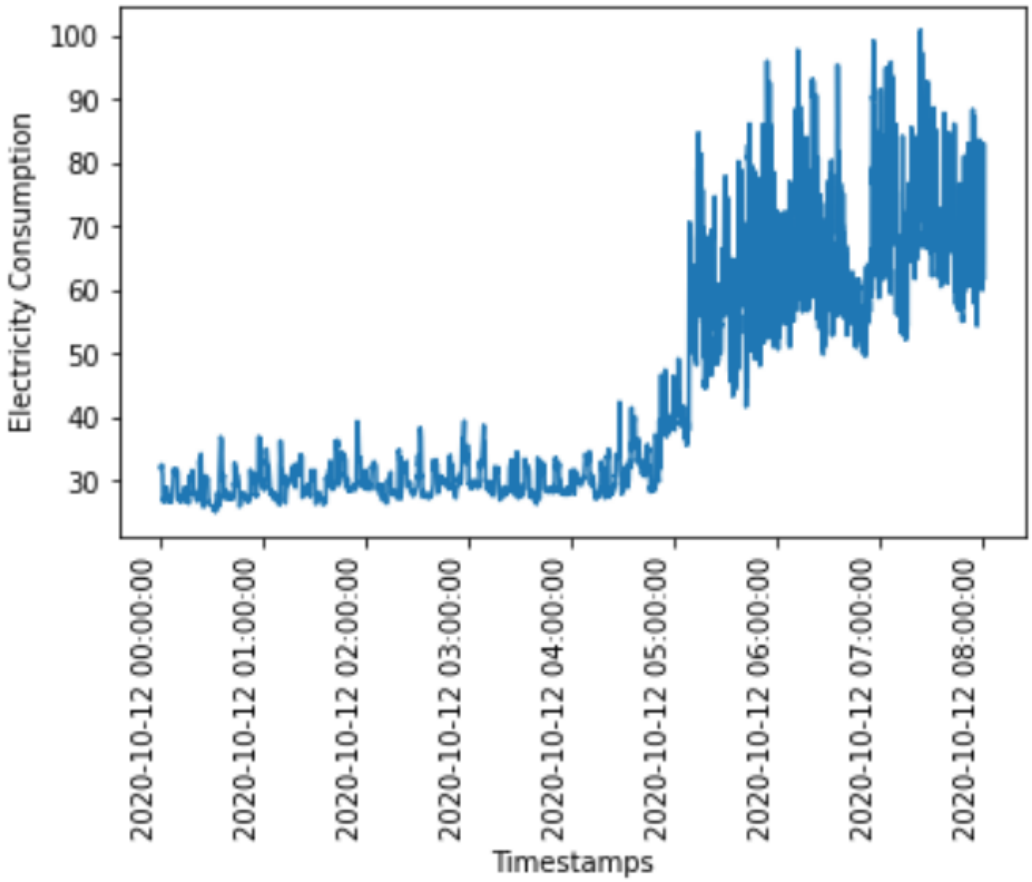


Figure 4. Electricity Usage Level During 8 hours of a Day in Original Dataset

It was observed that the synthetic data generated using CTGAN did not actually behave similarly to the original electricity usage level during the 8 hours of the day, rather the generated data was somewhat random. Also, the disadvantage was the model could not be fitted for larger datasets but usually, time-series datasets usually tend to be larger. Though CTGAN is able to generate new timestamps and their corresponding value of data, the generation does not happen in real-time and thus is not suitable for IoT device emulation.

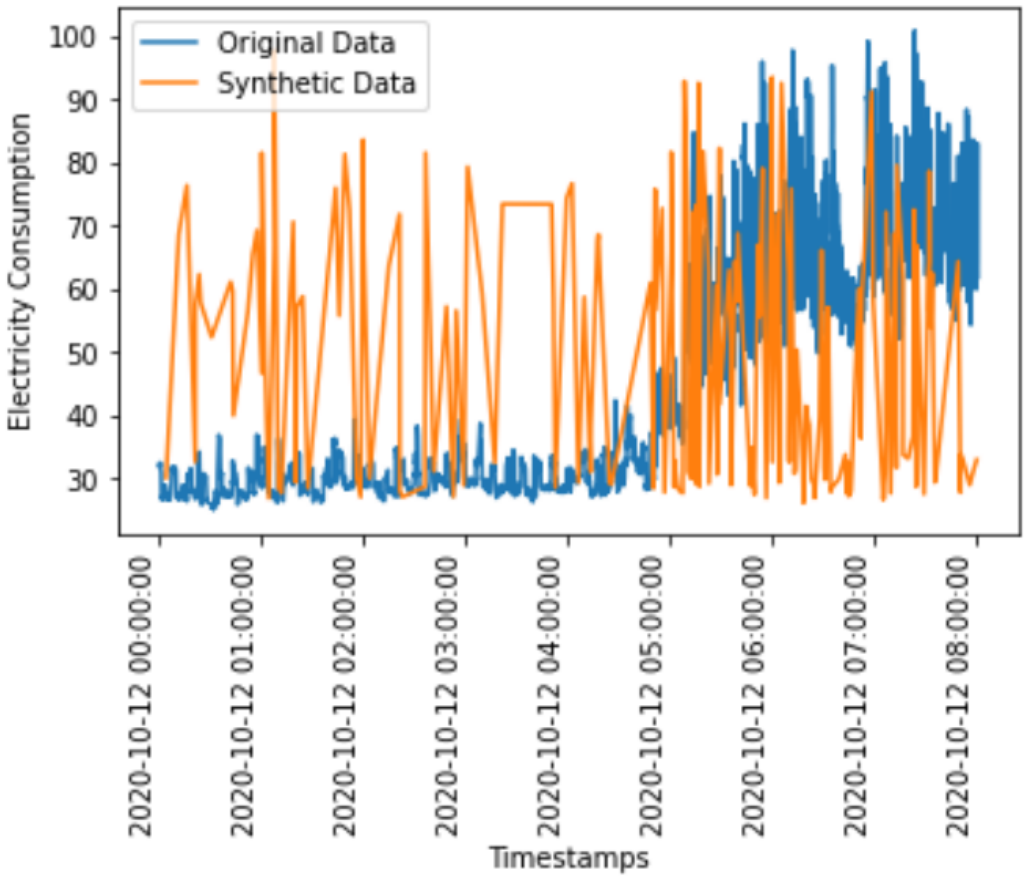


Figure 5. Comparison of Synthetic Electricity Usage Level Generated using CTGAN with Original Data

3.3 Synsys

SynSys is a system written in Python 3.6 used to generate synthetic sensor event data using a real smart home data file as a basis. SynSys uses Hidden Markov Models (HMMs) to generate activity and sensor event sequences and regression to generate timestamps. The steps to use this data generator were simple and well documented. Only a python command with certain parameters is sufficient to use Synsys and generate synthetic time series data. It accepts an existing dataset as input and is expected to generate similar behaving data like that of the actual dataset. However, it does not support datasets in any

format, rather expects the input dataset to be organized in a certain way. The columns in the dataset must be in the format as follows: **<date> <time> <sensor name> <sensor state or reading> <activity label>**.
The data generator was experimented with using the University of Tartu Electricity Consumption dataset. Initially, the dataset had to be pre-processed before using. The dataset consisted of the columns; date, time, sensorID, sensor value but in order to match the format of the input of Synsys, an activity label was to be added. Hence some random activity labels were added to the sensor data having different sensorIDs.

The python command which is used to run Synsys has the following format:
python runSynSys.py <path to dataset> <integer number of HMM states for activity HMM> <integer number of HMM states for sensor event HMM's> <integer time interval in hours for timestamp reset> <integer to indicated type of model for timestamp generation>
Hence, the Electricity Consumption Dataset was used as input and Synsys was run using the command:

```
python runSynSys.py dataset.csv 10 10 6 1
```

Observing the figure 6, it somewhat seemed like that Synsys could generate similar time-series data like the actual dataset but no definite conclusions could be made. However, a major drawback of this data generator was that, it could not be controlled how many samples has to be created. It was also hard to determine if the data generator was actually working as expected since no clear demonstration of examples using Synsys was available. The source codes were quite old and involved installing many old versions of different packages. Hence, it was quite difficult to get it to work for experimenting. Even though it generates synthetic time series data, it cannot be confirmed that the synthetic data possesses seasonality like the actual data, and also the synthetic data is not generated in real-time making it less suitable for IoT device emulation.

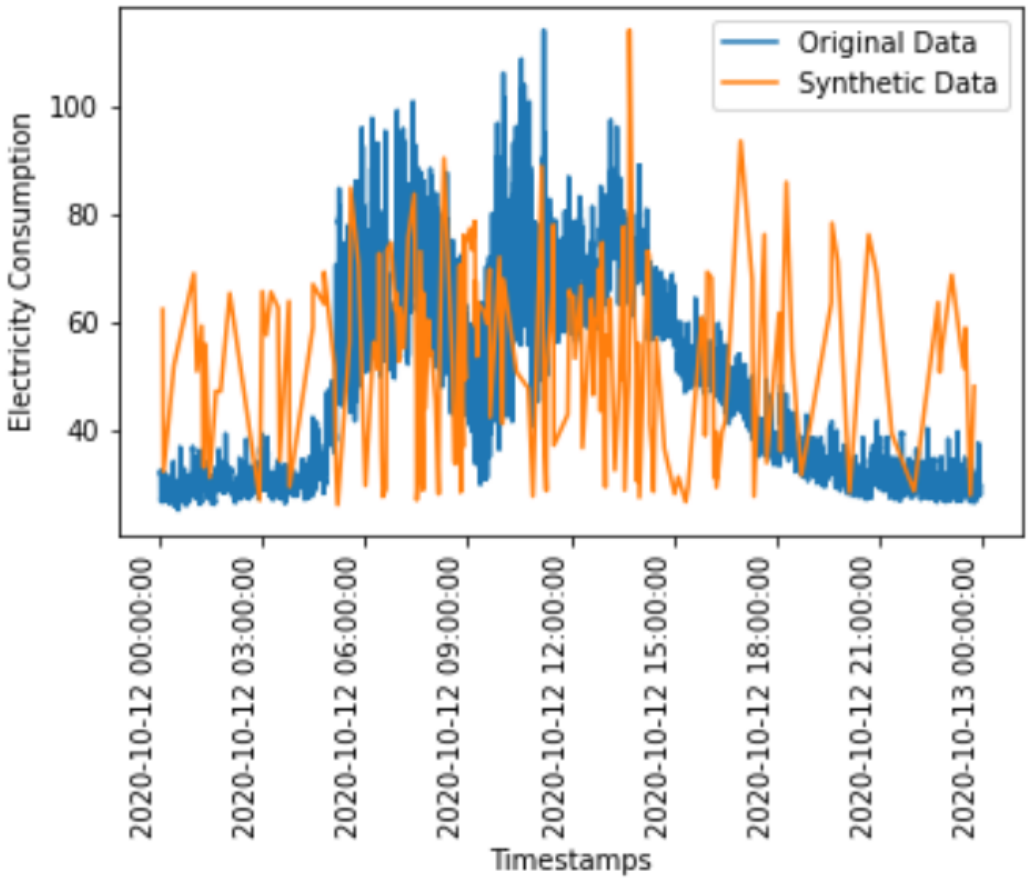


Figure 6. Comparison of Original and Synthetic Electricity Consumption Data Generated by Synsys

3.4 log-synth

log-synth is a data generator that can build fairly realistic files for analysis. The primary purpose of log-synth is to generate data in accordance with a given schema. Log-synth provides the functionality of generating data based on a schema. This schema-driven generation is capable of generating unique IDs, random integers, float or strings, and data with a defined distribution. In order to generate data using log-synth, some arguments need to be provided such as:

- The number of samples of data are to be generated
- The schema file is in JSON format and consists of a list of field specifications. Each field specification is a JSON object and is required to have the following value

The arguments are provided through the command line in the following format:

```
log-synth -count [number of samples] -schema [schema file]
```

In order to experiment with log-synth using the Electricity Consumption dataset, it was necessary to create a schema from the dataset which could be used to run log-synth and generate synthetic data. The distribution in which the data has to be generated is specified in the schema. Since, log-synth only has a few implementations so distributions(normal and random walk) in its application, there are not many options to use as the "class" parameter in the schema. Here, the mean and standard deviation of the overall electricity consumption data for a working day was computed and used as parameters in the schema. The schema used for the experiment was as follows:

```
[
    {
  "class":"sequence",
  "length": 3,
  "base":{
      "class":"map",
      "value": [
    {
        "name": "sample-time",
        "class": "event",
        "format":"yyyy-MM-dd HH:mm:ss",
        "start": "2020-10-01 1:00:00",
                    "rate": "1/h"
    },
    {
        "name": "electricity",
        "class": "random-walk",
```

```
            "mean": 53.008,
            "sd": 15.210,
            "start": 33.70
        }
          ]
    },
    "flat": true
      }
]
```

The schema file is prepared using the statistic of the data in the original dataset. Then, using the command to start log-synth synthetic data is generated for the electricity consumption measurements. It was observed that the generated data was somewhat random with similar range and mean as of the original data. Since the data was ransom and not bearing the patterns of the original data, it looked similar to the plot shown in figure 3. Though log-synth works well for generating random data with defined distributions and ranges in real-time, it is difficult to generate data that behaves like real-world IoT devices because it lacks the ability to demonstrate seasonal and observable patterns. Additionally, the data produced cannot be directly transmitted to any target systems such as MQTT or Kafka, as required by the CIISIM project.

3.5 Iosynth

Iosynth is an IoT device simulator as well as a synthetic data generator. The objective of this simulator is to emulate real-world IoT devices by generating data with a given frequency and distribution. Additionally, Iosynth supports MQTT and RabbitMQ (AMQP) protocols, which simplifies the process of sending provided data to these target systems. To begin using Iosynth, the maven package is created, which results in the creation of a target folder containing jar files with required dependencies. A single Java command is sufficient to start the simulator, which requires two configuration files as parameters: The MQTT client information configuration file and the device configuration file. The

table 1describes the parameters used in the MQTT client information configuration file.

Parameter	**Description**
uri	tcp://host[:port] - MQTT broker address
topic	Prefix of the MQTT topic that together with the device topic create the full topic name
clients	Number of MQTT clients. Each client sends the data from selected devices only.
qos	Quality of Service: 0, 1, 2
seed	Random Generator seed used to create reproducible scenarios. It can be omitted in other cases.

Table 1. Parameters for MQTT Client Information Configuration File

Parameter	**Description**
type	Device type
uuid, ipv4, mac48, mac64	Device uuid
topic	String forming the MQTT topic name
sampling	Device data sampling interval. Time parameters are in milliseconds.
copy	number of replicas for each device.
out_of_order	Double number [0..1.0] that sets the probability for out-of-order messages. (optional)
message_loss	Double number [0..1.0] that sets the probability for message loss. (optional)
sensors	list of sensor definitions.

Table 2. Parameters for Device Configuration File

The device configuration file is essentially a JSON template that specifies the parameters (Table 2) of the devices, including the type of data that must be produced by the devices. Additionally, the frequency of data generation can be defined here.

The Iosynth data generator was experimented with using the electricity consumption dataset. Iosynth is similar to log-synth in terms of functionality, but it allows for more flexibility in terms of defining the type of data to be generated and also has more implementations of distributions to choose from. Additionally, it has the capability of producing synthetic data for multiple devices with unique identifiers rather than just one. The format of the MQTT client information configuration file is as follows:

```
{
  "uri": "tcp://localhost",
  "topic": "iosynth/",
  "qos": 2,
  "clients": 1,
  "seed": 123456
}
```

The MQTT client configuration file specifies the MQTT broker address as $localhost$ and the topic format to be started with $iosynth/$.

```
[
    {
        "type":"simple",
        "uuid":"dev%04d",
        "topic":"device/{$uuid}",
        "sampling":{"type":"fixed", "interval":3600000},
        "copy":1,
        "sensors":[
            {"type":"dev.uuid",          "name":"uuid"},
            {"type":"dev.timestamp",     "name":"ts"},
```

```
        {"type":"double_walk", "name":"electricity", "min":33.70, "max
            ":114.137}
          ]
    }
]
```

The above device configuration file configures the type of data to be generated. Random UUID for the device is generated and concatenated with the MQTT topic $'device/[uuid]'$. Hence the generated synthetic data for each device can be obtained by subscribing to the topic $'iosynth/device/[uuid]'$. To replicate the behavior of the data in the electricity consumption dataset, the electricity usage value is created using the $double_walk$ generator, which generates random electricity usage levels within a defined range determined by the dataset's analysis. Since the data was ransom and not bearing the patterns of the original data, it looked similar to the plot shown in figure 3.Iosynth is capable of producing random IoT data within a defined range and displaying similar statistical metrics such as minimum, average, and maximum values as found in the actual dataset. However, it was limited in its ability to generate observable patterns or spikes in data from real-world IoT devices. One more advantage of Iosynth was that it could be used to send the produced easily to an MQTT broker, which met the CIISIM project's requirement.

3.6 Conclusion from the Experiments

From the experiments, it was concluded that, the existing data generators experimented with did not completely fulfill the requirements of the CISIIM project mentioned in section 1.1. But a suitable candidate can be selected which can be reused and made to adapt with the requirement of generating synthetic sensor data in real-time with similar behaving patterns. Following a thorough analysis of the available data generators, the IoT simulator Iosynth was chosen and we decided to customize it in order to adapt it for achieving the desired results. The choice to modify Iosynth was made for the following reasons:

- Features of Iosynth such as configuring device profiles, sending generated data to MQTT broker, customizable frequency, and generation of data with different distributions matched the closest to the project requirements
- It is easy to configure and conduct experiments
- Possible to deploy in Docker container
- Proper documentation and source codes are available
- Implementation is simple and more convenient to customize.
- Possessed the functionality of having everything configuration in files

4 Proposed Approach for Synthetic IoT Data Generation

According to the experiments and analysis of various data generation tools and simulators mentioned in the previous section, no suitable data generator could be identified that directly met the requirements of the CIISIM project. However, all of the data generators possessed some functionalities that allowed them to meet the requirements to a certain extent. It was observed that while it was simple to produce synthetic data with random behavior using existing data generators, it was difficult to emulate the seasonality of IoT devices at specific time intervals. For instance, if an IoT device generates data on the luminosity of the outside environment over the course of a day, it is assumed that these data do not exhibit random behavior but rather follow a noticeable pattern. In this case, the luminosity values will be near zero throughout the night, steadily rising to peak levels at midday, and then gradually decreasing to zero again at sunset. Another example could be the measurement of the electricity consumption of a smart building. It is common to observe an almost similar pattern of electricity usage during working days and a different pattern on the weekends. Such weekly behavioral change was also not possible to replicate using the data generators discussed so far. Replicating these forms of behavior was the main challenge to overcome. Hence, we decided to reuse the existing data generator Iosynth based on the reasons stated earlier in section 3.6 and build a solution around it.

It was obvious that an effective solution was needed to meet the previously stated requirements. Hence, a solution to approach this problem was designed. The goal of the solution is to generate data that would behave similarly or exhibit similar characteristics to that of the original data in a dataset. The original dataset contains IoT data, to be more specific sensor data from IoT devices (typically the sensors used in smart cities or buildings). Initially, the original dataset is pre-processed which involves removing outliers. The data is then plotted against their timestamps to reveal any noticeable behavioral pattern. It might be not convenient to find out such a pattern from the whole data. Under such circumstances, the time period of the data is decreased and plotted every time until a repetitive pattern is observed. In the case of replicating the pattern of a single

day, the whole day is divided into intervals each of an hour. Then each of the intervals is analyzed to find out the distribution of the data (random, normal, linear increasing or decreasing, etc) in that particular interval. This would create profiles of the IoT device at a particular interval of the day. At this point, the data generator Iosynth is used to utilize these device profiles in order to generate similar sensor data. For each interval, Iosynth is launched using a different device configuration file. The orchestration of selecting corresponding device configuration files for each interval is done using a Python Script. The Python script is executed per interval to launch Iosynth with a different configuration file. Execution of the Python script every interval is done using a $CRON$ scheduler. Combining the data generated per hour interval using these devices profiles, it will be possible to have synthetic sensor data over the period of a day. Iosynth generates data in real-time, and so it behaves just like a virtual IoT device. Similarly, if it is required to have different patterns of behavior on different days (such as working days and holidays), then the device profiles for such days would be created separately in order to be emulated.

4.1 Architectural Workflow of the Proposed Approach

The Python script (Orchestrator), the data generator Iosynth, the CRON Scheduler, and the MQTT client are the key software components of the proposed solution, as detailed above. Figure 19 depicts the architectural workflow of the proposed solution. The overall summary for synthetic data generated using the proposed solution is as follows:

1. The dataset which has to be replicated is analyzed to ascertain the correlation and behavioral pattern among the data. Also, the dataset is divided into smaller intervals to determine the distribution of the data in each interval. Additionally, statistical details of the data per interval are be obtained, such as the data's minimum or maximum range, mean, deviations, etc.

2. The device configuration files are created for every interval based on the analysis from the previous step. It holds information such as the frequency in which data

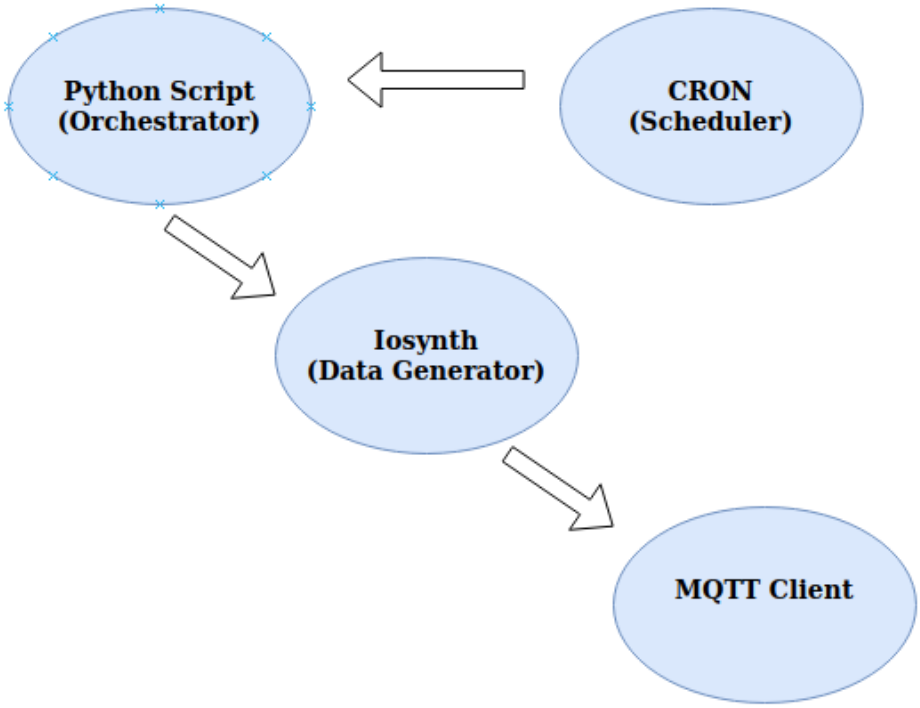


Figure 7. Key Software Components of Proposed Solution

should be produced, the number of devices for which data should be generated, distribution of the data, and other necessary parameters.

3. The Python script acts as the orchestrator which handles the logic for selecting which device configuration file to use at a particular interval and also invokes the Iosynth simulator by using the prepared device configuration files and passing them as a parameter to the Java command that launches Iosynth.

4. A Cron job is scheduled to run the python script at defined intervals and invoking the Iosynth simulator with appropriate device configuration files, thus emulating the behavior of real IoT devices.

5. Data is generated by the Iosynth simulator following all the specifications made in

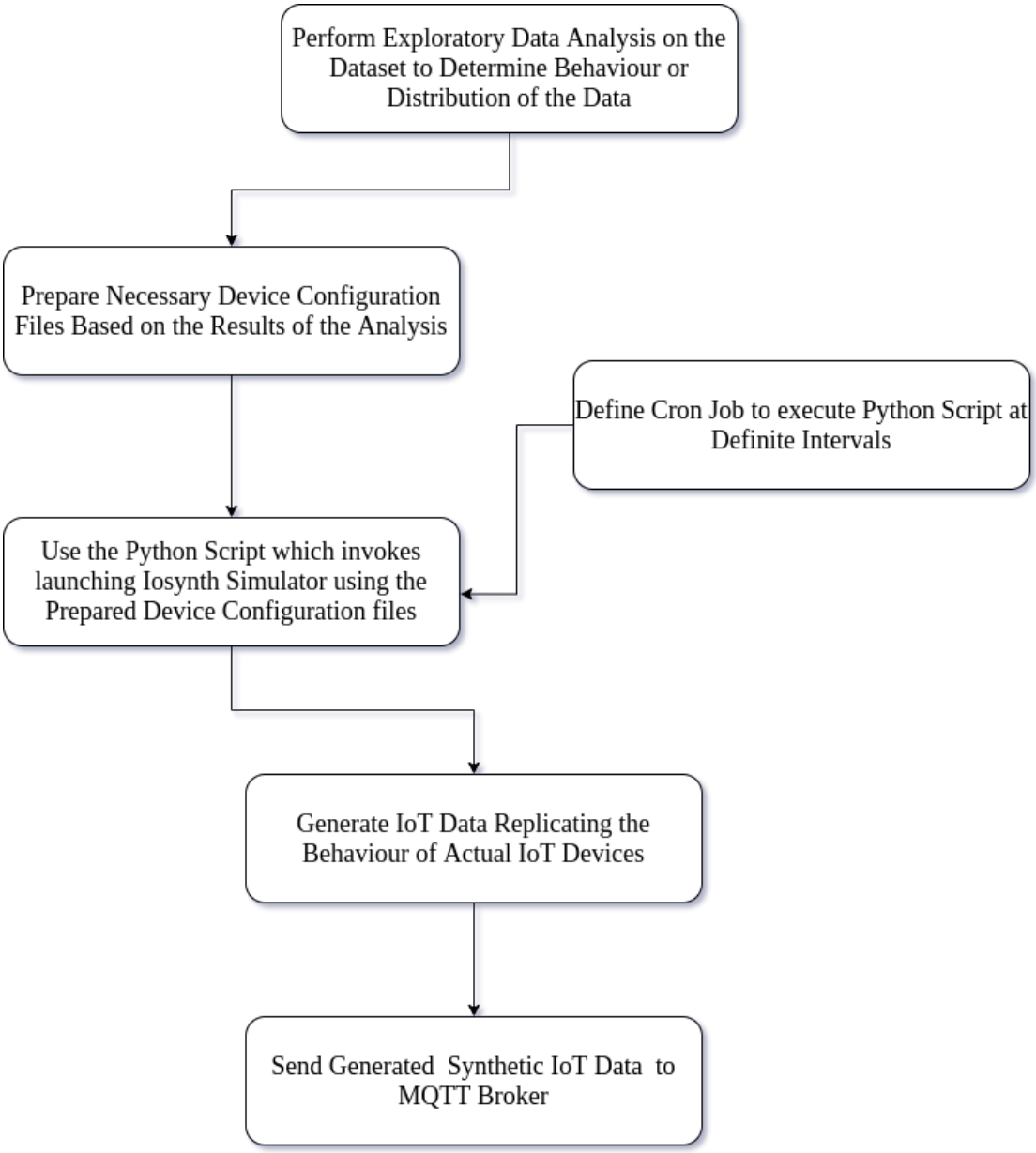


Figure 8. Architectural Workflow of Proposed Solution

the second step.

6. These synthetic IoT Data after generation are sent to the MQTT client which was specified in the MQTT client information configuration file.

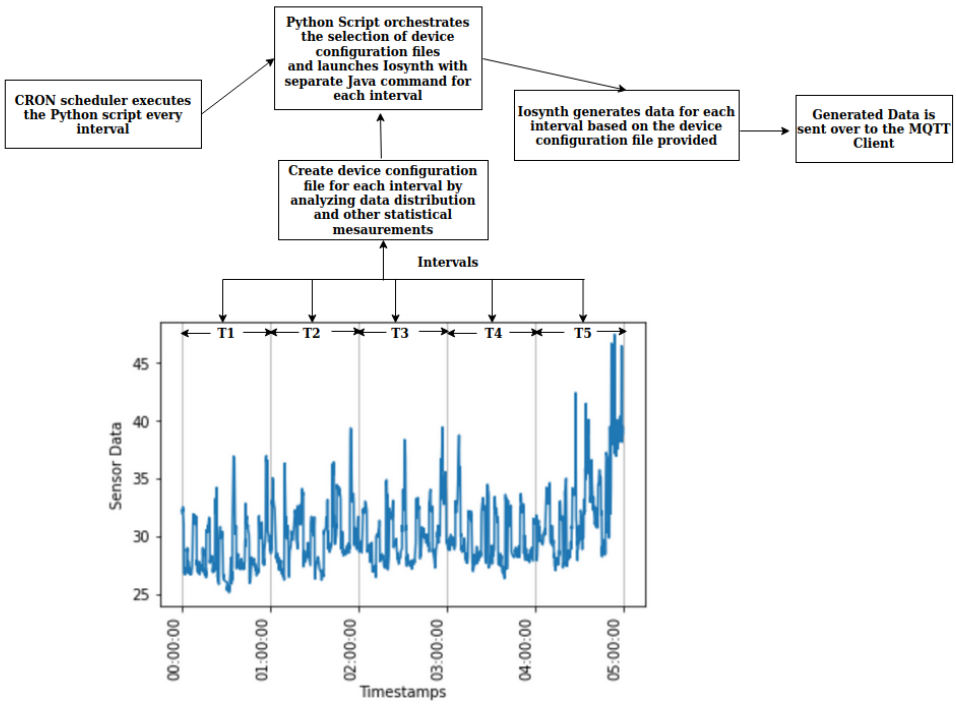


Figure 9. Illustration of Proposed Solution

4.1.1 Analysis of Dataset and Creation of Device Profiles

The first step of the solution is to analyze the original dataset and remove outliers if any. The dataset is graphically plotted to visualize the behavioral patterns. As mentioned earlier, the data is divided into smaller intervals to determine the behavioral pattern of the data for each interval. In the implementation, an interval of 1 hour is used. For each day, the whole data of a particular day is divided into 24 intervals, each of 1 hour. Later determination of the data distribution (random, normal, linear increasing or decreasing, etc.) is done for each interval (1 hour). In this manner for each interval, a profile of the IoT device is created and saved in JSON format which contains information on the distribution of the data, how frequently the data is generated, and other necessary parameters like minimum, maximum or average values, standard deviations, anomaly,

etc. Once the device profiles have been created for each interval, we will have a set for a single day that would be used by the data generator to generate similar behaving sensor data. The overall data pre-processing and creation of device profile JSON files are implemented using Python.

4.1.2 Execution and Scheduling of the Orchestrator

The existing data generator, Iosynth currently supports the use of only one device configuration file which means it is not possible to use different device configuration files for different time intervals. To emulate the behavior of real IoT devices based on other factors such as the time of day, month, or season the Iosynth had to be adapted to use different device configuration files having varying configurations. The proposed solution includes adapting Iosynth to use multiple device configuration files rather than a single one, generating data with varying configurations depending on other conditions. The Python script handles the logic associated with the selection of device configuration files to be passed as a parameter to the Java command that launches Iosynth. In order to execute the Python script properly, a device configuration selection file in JSON format is created which contains the names of the device configuration files with the time intervals of a day as keys. The Python script reads the device configuration selection file, fetches the current timestamp of the day, and determines which device configuration file has to be used by Iosynth to generate synthetic data for that particular time interval. Execution of the python script is done at predefined intervals, each time varying the device configuration file and passing it as the parameter in the Java command by using CRON scheduler. CRON is a time-based job scheduler used in Unix-like computer operating systems. For cron jobs to run, mainly four components are required:

- The script which is to be called or executed
- Schedule of the script to be executed
- The command which executes the script

Character	Descriptor	Acceptable values
1	Minute	0 to 59, or * (no specific value)
2	Hour	0 to 23, or * for any value. All times UTC.
3	Day of the month	1 to 31, or * (no specific value)
4	Month	1 to 12, or * (no specific value)
5	Day of the week	0 to 7 (0 and 7 both represent Sunday), or * (no specific value)

Table 3. Cron Job Parameters

- The output or action of the script that is determined by the function of the script being invoked.

The cron job is specified in the following format shown below where the first 5 parameters are described in Table 3.

```
* * * * * [command to run script] [script to be executed]
```

As previously stated, the Iosynth simulator is initiated through a Java command that creates a java process for the simulator to run and generate data. However, it had to be ensured that repeating the java command with different device configuration files would not result in a large number of java processes running concurrently, as this would cause issues with maintaining behavioral patterns during data generation. This issue has also been solved in the Python script by ensuring execution of the script happens only after all previous Java processes involving running Iosynth have been terminated.

4.2 Modifications Made in the Data Generator

As described earlier Iosynth requires a device configuration file where it can be specified what type of data has to be generated. Although the generation of data with different distributions was possible using Iosynth, it lacked the function to generate spikes and drop patterns or linearly increasing and decreasing behavior of data which was necessary in cases where such data like the example of luminosity level stated earlier needed to be replicated. Also, there may be cases where the original data does not necessarily resemble any known distribution. In that case, a separate Java class was introduced in Iosynth which would be able to replicate most distributions. Overall, 3 more extra Java classes were added to the distributions set of Iosynth. These were as follows:

- ***SensorDoubleIncrement.java***: This Java class when specified as a distribution in the device configuration file, generates random values in a specific range in a linearly increasing manner. It expects a lower and upper limit and also a slope which determines the increase between the previous and next data to be generated as arguments.

- ***SensorDoubleDecrement.java***: This Java class when specified as a distribution in the device configuration file, generates random values in a specific range in a linearly decreasing manner. It expects a lower and upper limit and also a slope which determines the decrease between the previous and next data to be generated as arguments.

- ***SensorDoubleMean.java***: This Java class expects a value a mean or average, an array of deviations, and an anomaly as arguments. The mean(single value) and deviations(array) from the mean are calculated from the original data and used as inputs to this distribution. The anomaly is the extra parameter that can be specified by the user, which makes the synthetically generated data different from the original one. Whenever this class is called, the mean value is added to a randomly picked deviation from the array list plus the anomaly.

4.3 Containerized Deployment

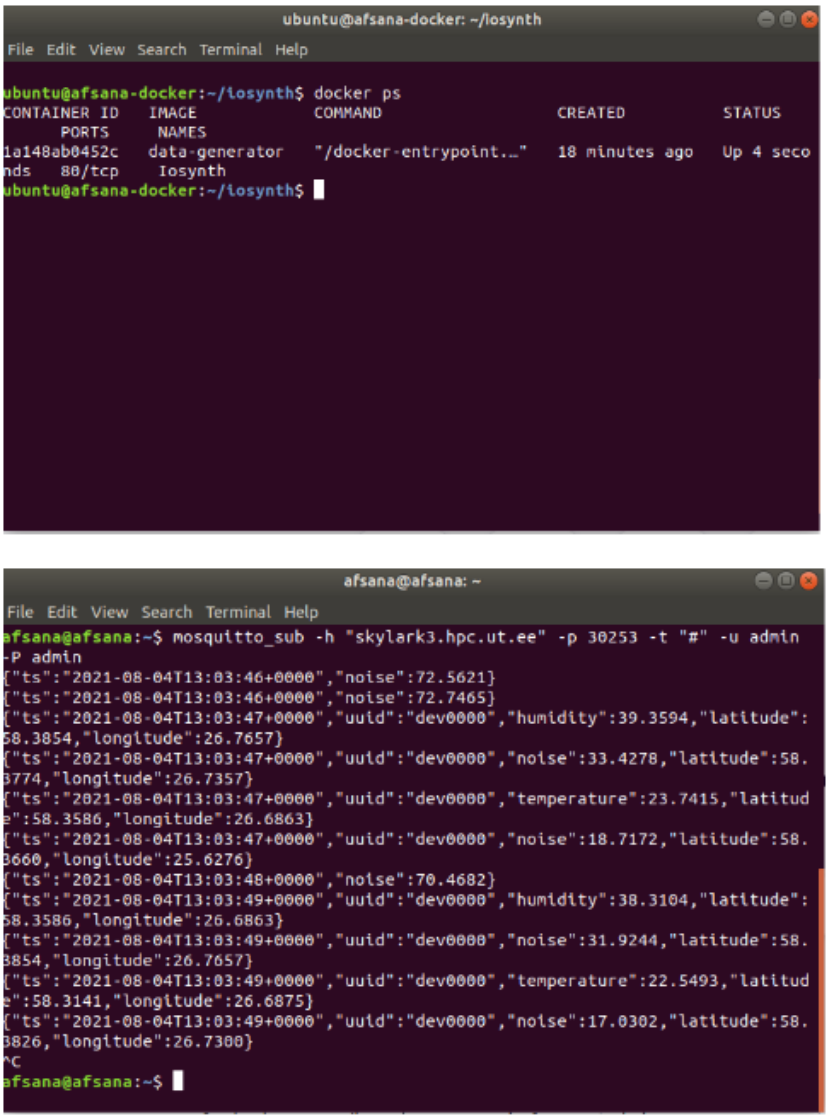


Figure 10. Containerized Deployment of the Data Generator

Each virtual IoT device is set up as a running docker container. A Dockerfile was created to install Iosynth with its required dependencies and parameter, installing python and executing the script with CRON. The docker image was build using the Dockerfile. The build image was used to tun the container. When the container starts running data is generated following the proposed approach and published to MQTT which can be viewed when subscribed.

5 Evaluation of the Proposed Approach

Since the project CISIIM involves the use cases of a smart city, it was felt that using data from sensors that are commonly used in smart cities or smart buildings, such as solar levels and electricity usage, would be a good option. The Delta Centre of the University of Tartu is a good example of a smart building and it was also possible to have direct access to the data. In order to evaluate the quality of the proposed approach, two datasets from the Delta Centre have been used.

5.1 Delta Centre Electricity Usage Dataset

The Dataset used for the evaluation purpose is the Electricity Usage Dataset of the Delta Centre. It contains information on the electricity usage level (in Kilowatts) throughout different days from $2020/09/23$ to $2020/11/23$ collected from the sensors installed at the Delta Centre. The contents of the dataset are described in Table 4.

timestamp	Time and date when the value was collected. Integer Format.
host	ID of the sensor
unit	Unit of the value
value	Electricity Usage Level (KW)

Table 4. University of Tartu Delta Centre Electricity Usage Dataset

Firstly, the dataset was pre-processed to remove outliers. The electricity usage levels were plotted against their timestamps for a period of one month. Figure 11 shows the electricity usage pattern of the Delta Centre for a period of one month. Although a noticeable pattern can be observed, it is not yet possible to determine the exact behavior. Hence, the data is again plotted in a smaller time period of one week as shown in Figure 12.

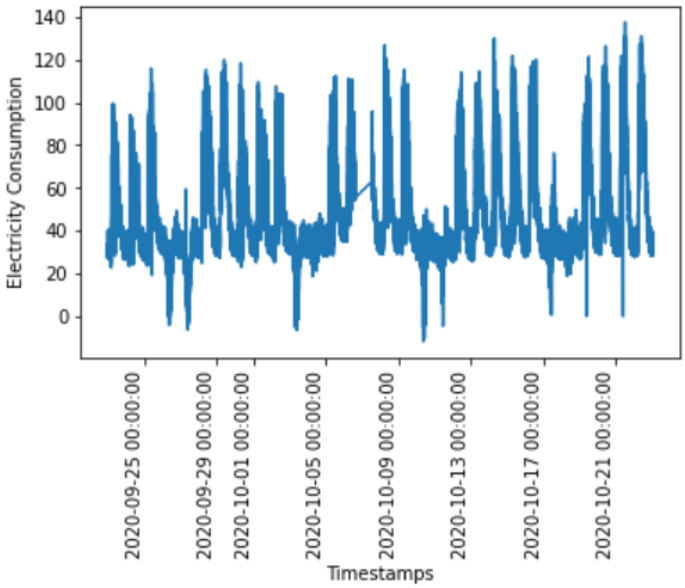


Figure 11. Plot of Electricity Usage Level Data for a Month

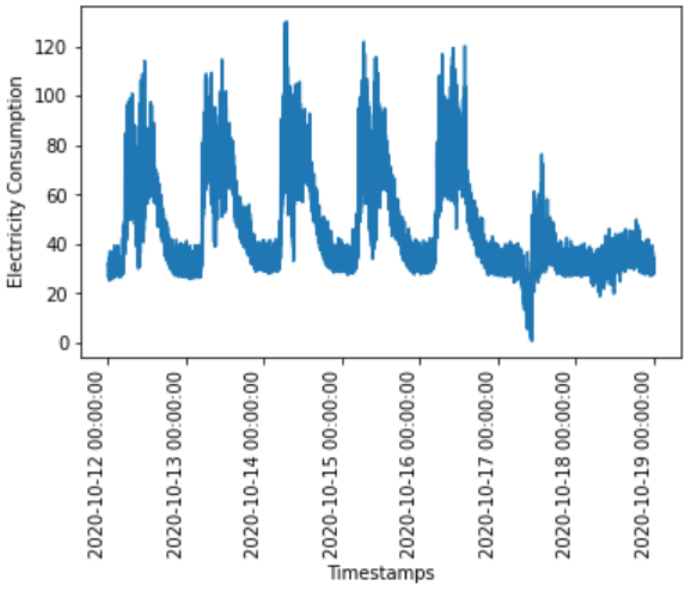


Figure 12. Plot of Electricity Usage Level Data for a Week

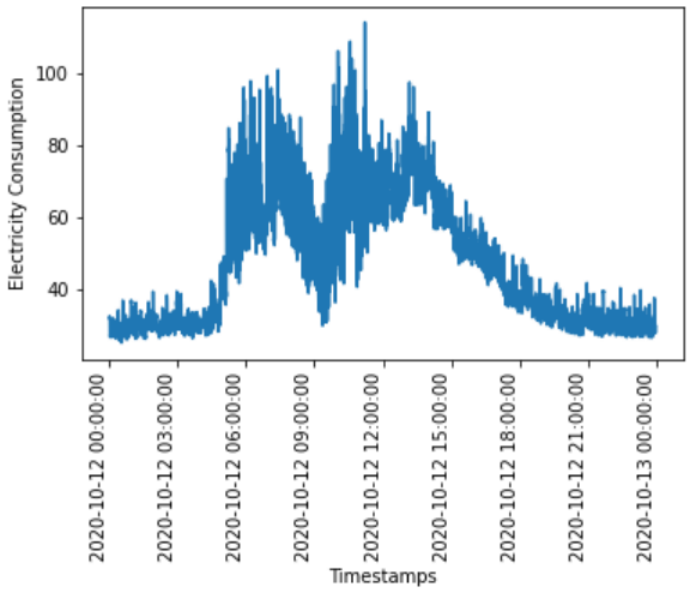


Figure 13. Plot of Electricity Usage Level Data During a Working Day

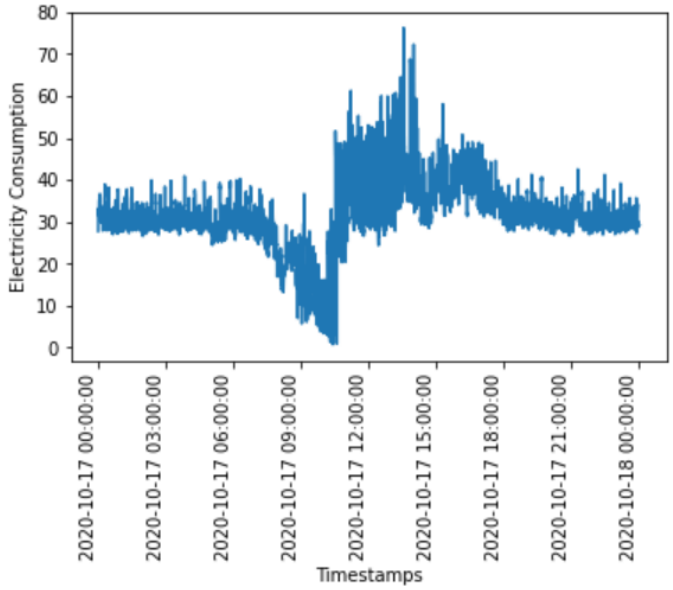


Figure 14. Plot of Electricity Usage Level Data During a Weekend

From the weekly electricity consumption plot a clear pattern is observed, i.e. the electricity usage levels during the working days remain almost the same. The usage pattern seems to be somewhat similar in the working days but there is a drastic difference in the case of weekends. Again during both the days of the weekend (Saturday and Sunday), similar usage patterns can be seen.

Now, this behavior or electricity consumption can be replicated using the proposed approach. Since the pattern is similar for all the working days and a similar pattern is observed on the weekends, any particular working day and weekend are selected to generated device profiles. In the case of a working day, the whole day is divided into time intervals of 1 hour and therefore into 24 intervals. For each interval, the mean (average) of the electricity consumption (Kilowatts) and as well as an array consisting of the deviations of the values from the mean is computed. The device profile file (JSON) will contain the mean and array of deviations as parameters. While generating synthetic data the data generator (Iosynth), the mean value is added to any randomly selected deviation from the array, and also an anomaly value is added so that the synthetic data differs from the original data. The default anomaly value is set to 10 which can be varied by the user. For each time interval, a device profile or device configuration file is created and stored in a directory. The same procedure is followed in case of a day on the weekend and device configuration files are stored in another directory. A template of the device profile or configuration file is as shown below:

```
{
        "type":"simple",
        "uuid":"dev%04d",
        "topic":"device/{$uuid}",
        "sampling":{"type":"fixed", "interval":N/3600000},
        "copy":1,
        "sensors":[
            {"type":"dev.timestamp",     "name":"ts"},
            {"type":"dev.uuid",          "name":"uuid"},
            {"type":"double_mean", "name":"electricity_usage", mean:
                M, std: [d1,d2,d2.....dN], anomaly: A}

        ]
```

```
}
```

Listing 1. Template Device Configuration File

Here N = number of original values in a particular time interval, M = mean of the values at a particular interval, A = anomaly. Once the device configuration files are created, then these are used as parameters by the data generator Iosynth. Since we will be using different device configuration files for different days and time intervals, the python script handles that logic. Execution of python script each time launches the Iosynth data generator to run with a particular device configuration file. To execute the python script at a regular one-hour interval, the $CRON$ scheduler is used.

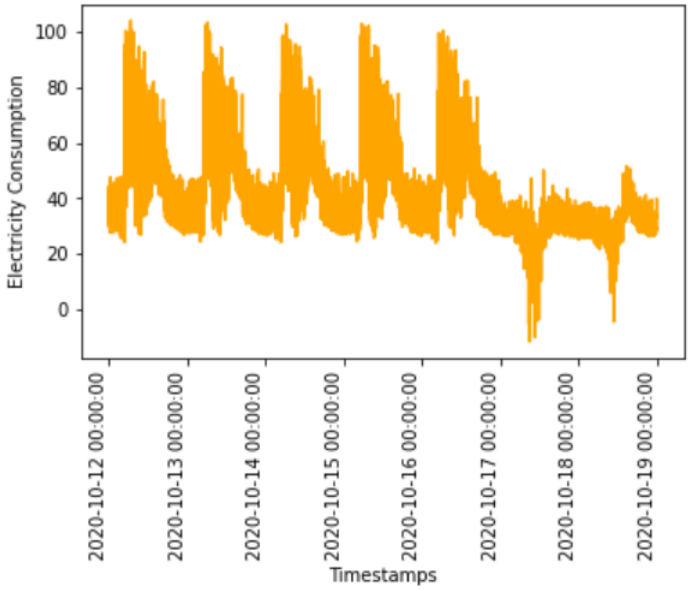


Figure 15. Plot of Synthetically Generated Electricity Usage Level Data for a Week

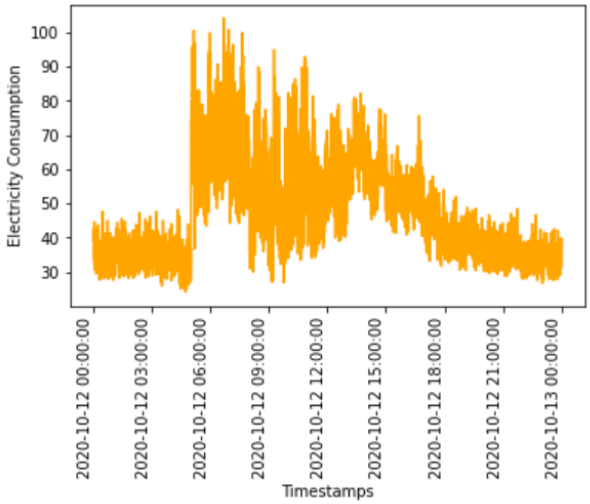


Figure 16. Plot of Synthetically Generated Electricity Usage Level Data During a Working Day

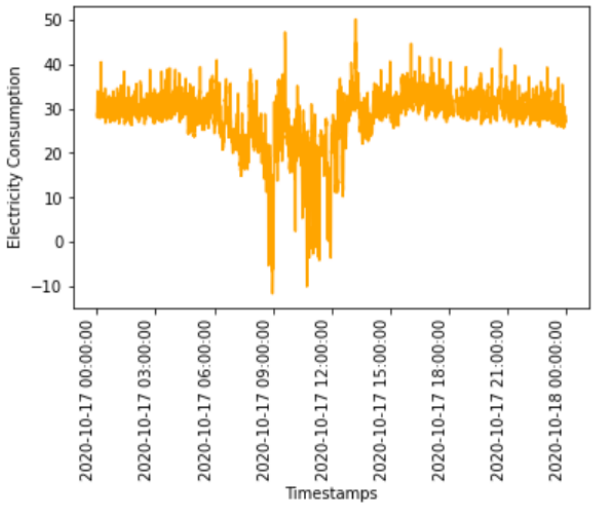


Figure 17. Plot of Synthetically Generated Electricity Usage Level Data During a Weekend

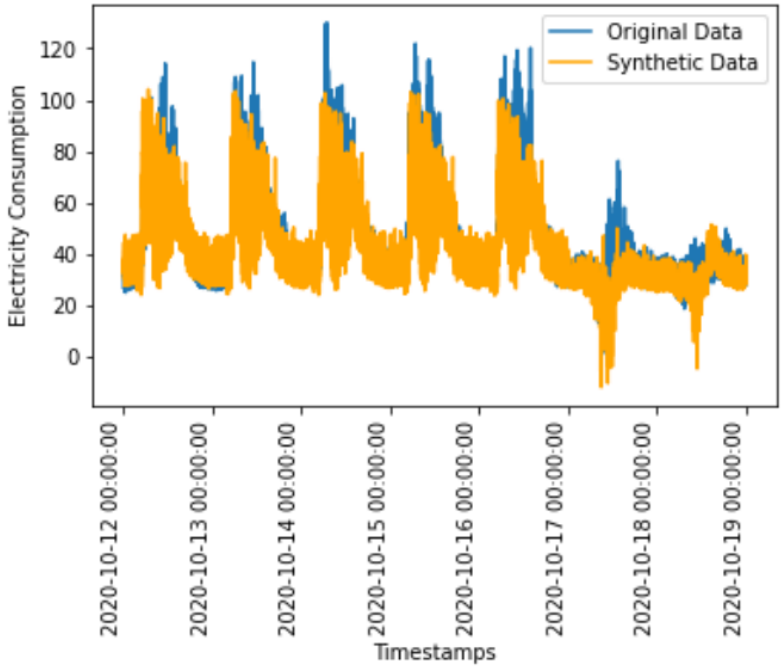


Figure 18. Comparison Between Original and Synthetically Generated Electricity Consumption Data

The data was generated in real-time using the proposed approach. The generated data were sent over MQTT client from where they were collected and analyzed to compare with the original dataset. From the Figures 15,16 and 17, it can be observed that the synthetically generated data behaves almost similar to the original data, since the similar electricity usage patterns can be observed during the working days and change over the weekends. Thus, we were able to achieve the goal.

5.2 Delta Centre Solar Level Dataset

The University of Tartu Delta Centre Solar Level Dataset has been used where the objective is to generate similar behaving solar level data as that in the dataset. This dataset can also be seen as representing IoT devices that measure the outside light level.

For example, devices are used to optimize the energy production of solar panels. or Smart streetlights, which use the light level to decide when to turn on street lights and how strong they should be currently. The contents of the dataset are described in Table 5.

timestamp	Time and date when the value was collected. Integer Format.
host	ID of the sensor
unit	Unit of the value
value	Luminosity level outside

Table 5. University of Tartu Delta Centre Solar Level Dataset

A similar approach is performed on the solar level dataset as like as the one performed in the Electricity usage dataset earlier. The dataset was investigated initially, and necessary pre-processing was done to eliminate anomalous and missing values. Additionally, it was determined if there were any missing days within the specified time period during which data was not available. In the next step, the data was plotted against their timestamps to determine any repetitive pattern.

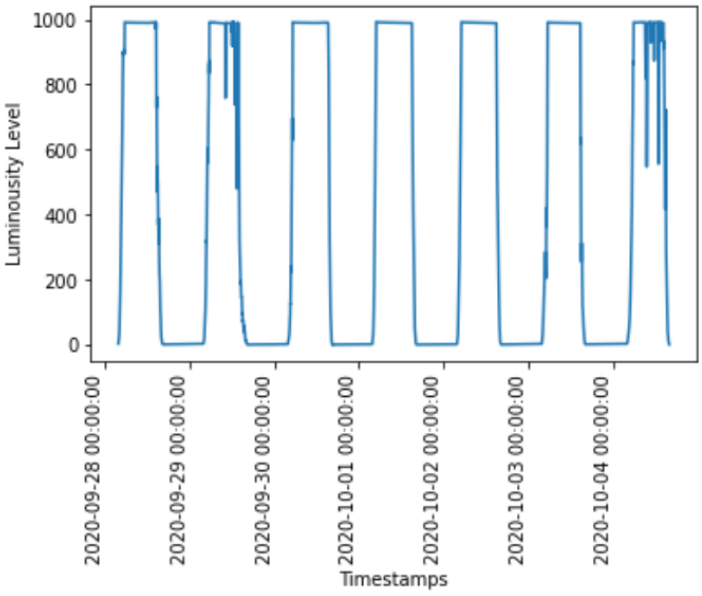


Figure 19. Plot of Luminosity Level Data for a Week

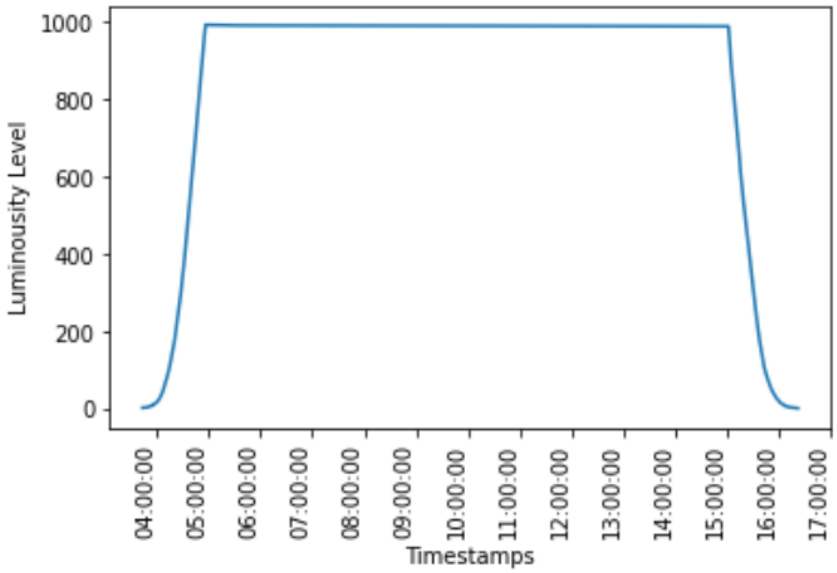


Figure 20. Plot of Luminosity Level Data for a Typical Day

From the figures 19 and 20, one can get an idea about the behavioral pattern of the luminosity level in a usual day. According to the graph is figure 20 where the luminosity levels have been plotted along the Y-axis against the timestamps in the X-axis, it can be observed that the luminosity levels are very low; almost tends to zero during the evening and night (approximately after 16:00 and before 3:00), the levels start to increase linearly when the sun rises (approximately from 4:00) which reaches the maximum value (around 1000 Lux) during mid-day, Finally starts to decrease linearly after 16:00.

To replicate the behavior as discussed above using the Iosynth simulator, the dataset was explored and analyzed. The analysis was done by plotting luminosity levels against the timestamps over a time period of a day or week. The trends such as the rise and fall of the luminosity levels were identified. A typical day was considered as an ideal time period and thus divided into smaller intervals each of 1 hour. The distribution of the data and as well as other measurements were computed for every interval. Using this information for every hour or interval device configuration files were generated and stored. For example, the distribution of the data was linearly increasing from 3:00

to 4:00, almost constant within a range during the midday after 5:00 and gradually decreasing again after 16:00. The range(maximum and minimum value) for each interval was also determined and used as parameters in the device configuration file. The python script manages the selection of these device configuration files per interval and invokes the Iosynth data generator to makes use of these device configuration files to generate synthetic luminosity data with a pattern close to that of the actual dataset. A Cron job was scheduled to execute the python script every hour which invoked the Iosynth simulator using the device configuration files.

```
{
        "type":"simple",
        "uuid":"dev%04d",
        "topic":"device/{$uuid}",
        "sampling":{"type":"fixed", "interval":N/3600000},
        "copy":1,
        "sensors":[
            {"type":"dev.timestamp",     "name":"ts"},
            {"type":"dev.uuid",          "name":"uuid"},
            {"type": D, "name":"luminousity", "min":M1, "max":M2}

        ]
}
```

Listing 2. Template Configuration File for Generating Luminosity Data

Here D = {$double_increment$, $double_decrement$, $double_uniform$}, $M1$ = minimum value from the data in a interval and $M1$ = maximum value from the data in a interval. In the device configuration file used for emulating the luminosity data behavior during early morning, the data generator type $double_increment$ is used which is the modification made into the Iosynth simulator to generate values of data type double in a linearly increasing manner and the opposite is implemented in the type $double_decrement$ used in the device configuration file for generating luminosity data during the afternoon. During the mid-day or peak hours, the luminosity levels are near to

the maximum (1000 LUX) and do not fluctuate significantly. Hence for this case, the device configuration file used the type $double_uniform$ which would generate random values within a defined range near to the maximum value.

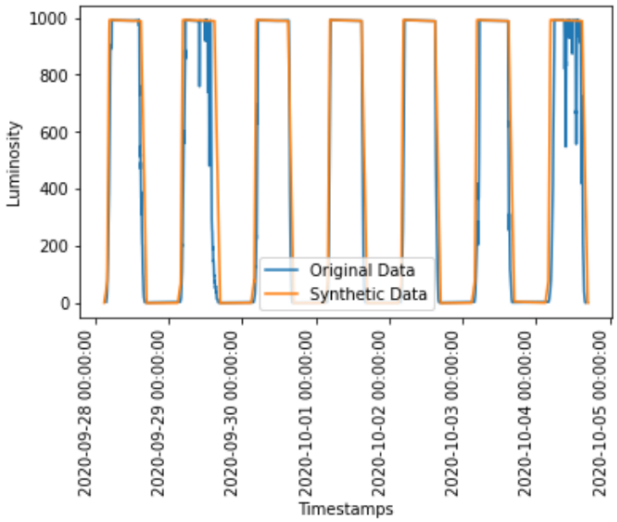


Figure 21. Plot of Synthetically Generated Luminosity Level Data for a Week

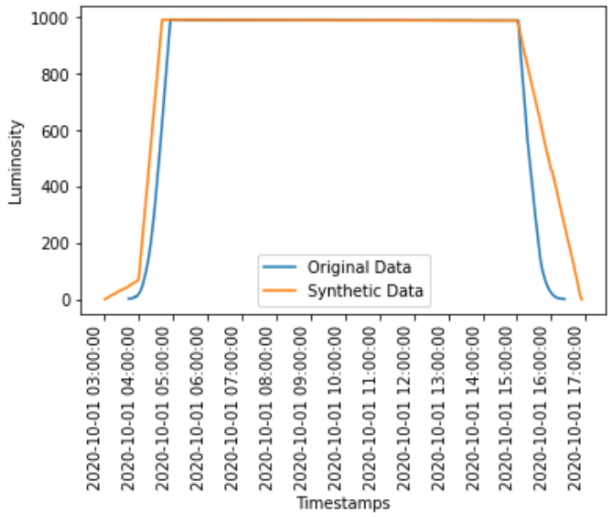


Figure 22. Plot of Synthetically Generated Luminosity Level Data for a Day

Hence from the figures 21 and 22 it can be observed that the synthetic luminosity data generated using the proposed approach almost behaves similarly to the original luminosity data in the solar level dataset.

6 Conclusion and Future Work

This thesis has focused on the challenges involved in emulating real IoT devices required for testing IoT systems. The objectives of the thesis included generating synthetic IoT data with customizable frequency that resembles real data generated by IoT devices, taking seasonality and other patterns into account. Thus, eliminating the need to have physically set up IoT devices for testing IoT systems. The accomplishments from this these are as follows:

- Experimental evaluation of existing data generators to investigate how suitable they are for generating synthetic IoT data that behaves similarly to original data, highlighting their benefits and disadvantages. This can be useful for users who need to choose which existing tools to use for their use cases
- A new approach was proposed for utilizing existing data generators to generate more elaborate synthetic data and which is able to switch between different data generation patterns at different periods
- A contribution to an existing open-source data generator by implementing new data generation distributions.
- Experimental validation of the proposed solution to demonstrate that it is able to mimic the patterns and behavior in typical IoT data.

The proposed solution was evaluated by using it on IoT datasets and comparing the generated synthetic data with the original one. The results plotted from the graphs showed that, the generated synthetic data possessed similar behaving characteristics and thus was able to emulate real IoT devices.

However, the proposed approach has a number of limitations that did not fit into the scope of this thesis and will be part of the future work. The future directions of the proposed approach are highlighted as follows:

- The proposed approach involves analysis of the dataset and creation of device configuration files. This step can be made more automated by adding the functionality of splitting the datasets into smaller sets (year, month, week, or day) and deciding on which data distribution resembles it most closely. In this way, manual plotting of the data of the dataset won't be required. Thus that data distribution decided automatically could be used as parameters in the device configuration file.

- The python script that orchestrates the selection of device configuration files and launching of iosynth could be adapted more to allow selection of device configuration files for data having seasonality changes within months or years. Currently, the selection of device configuration files can with data having to change patterns or trends within a day or week has been implemented.

- In addition, the proposed approach can be extended to adapt the generation of of IoT data based on actuator states and as well as generating synthetic but real-looking GPS coordinates. This is a requirement for the CIISIM project's Smart Scooter use case, which will be worked on in the near future. These modifications can be made by adapting the orchestrator python script.

In conclusion, the approach proposed in this thesis for generating synthetic IoT data that exhibits realistic behavior will be beneficial for industries where testing of IoT systems is conducted and require a large-scale real testbed. It would facilitate the testing procedure in cases where there aren't enough real IoT devices to connect. This approach has been used in the project CIISIM (1.1) which involves creating a Virtual village comprising of virtualized IoT devices. The virtual IoT devices were set up as docker containers and the extension of the data generator has been used inside them so that they can emulate data replicating the real IoT devices.

References

[1] J Rivera and L Goasduff. Gartner says a thirty-fold increase in internet-connected physical devices by 2020 will significantly alter how the supply chain operates. *Gartner*, 2014.

[2] Isakovic Haris, Vanja Bisanovic, Bernhard Wally, Thomas Rausch, Denise Ratasich, Schahram Dustdar, Gerti Kappel, and Radu Grosu. Sensyml: Simulation environment for large-scale iot applications. In *IECON 2019-45th Annual Conference of the IEEE Industrial Electronics Society*, volume 1, pages 3024–3030. IEEE, 2019.

[3] Saraju P Mohanty, Uma Choppali, and Elias Kougianos. Everything you wanted to know about smart cities: The internet of things is the backbone. *IEEE Consumer Electronics Magazine*, 5(3):60–70, 2016.

[4] Intel lab data. `http://db.csail.mit.edu/labdata/labdata.html`. (Accessed on 05/07/2021).

[5] Iot security. `https://iotanalytics.unsw.edu.au/`. (Accessed on 07/29/2021).

[6] Smartsantander. `https://smartsantander.eu/`. (Accessed on 07/29/2021).

[7] Rohyoung Myung, Sungmin Choi, Woosung Choi, Heonchang Yu, Daewon Lee, and Eunyoung Lee. Elaborate synthetic data generation for internet of things services at smart home environment. In *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 226–229. IEEE, 2016.

[8] J Zico Kolter and Matthew J Johnson. Redd: A public data set for energy disaggregation research. In *Workshop on data mining applications in sustainability (SIGKDD), San Diego, CA*, volume 25, pages 59–62, 2011.

[9] Asma Alotaibi and Charith Perera. Smart home human activity simulation tool for openhab-based research. 2019.

[10] Jason W Anderson, KE Kennedy, Linh B Ngo, Andre Luckow, and Amy W Apon. Synthetic data generation for the internet of things. In *2014 IEEE International Conference on Big Data (Big Data)*, pages 171–176. IEEE, 2014.

[11] Giuliano Casale and Ahmad Alnafessah. Qt: A quality testing tool for data-intensive applications.

[12] Son N Han, Gyu Myoung Lee, Noel Crespi, Kyongwoo Heo, Nguyen Van Luong, Mihaela Brut, and Patrick Gatellier. Dpwsim: A simulation toolkit for iot applications using devices profile for web services. In *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pages 544–547. IEEE, 2014.

[13] Ahcène Bounceur, Olivier Marc, Massinissa Lounis, Julien Soler, Laurent Clavier, Pierre Combeau, Rodolphe Vauzelle, Loic Lagadec, Reinhardt Euler, Madani Bezoui, et al. Cupcarbon-lab: An iot emulator. In *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pages 1–2. IEEE, 2018.

[14] Iba-group-it/iot-data-simulator: Generic iot data simulator. provides possibility to replay datasets or generates data on fly. supports various iot platforms out of the box. `https://github.com/IBA-Group-IT/IoT-data-simulator`. (Accessed on 05/02/2021).

[15] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Modeling tabular data using conditional gan. *arXiv preprint arXiv:1907.00503*, 2019.

[16] Jessamyn Dahmen and Diane Cook. Synsys: A synthetic data generation system for healthcare applications. *Sensors*, 19(5):1181, 2019.

[17] tdunning. log-synth, Aug 2021. [Online; accessed 4. Aug. 2021].

[18] rradev/iosynth: Iosynth is iot device/sensor simulator and synthetic data generator. `https://github.com/rradev/iosynth`. (Accessed on 05/02/2021).

Appendices

Appendix I - Repository for Source Codes

- The source codes for the extended version of Iosynth simulator are located in the Github repository *https://github.com/Afsana2910/iosynth*.
- Scripts to run the proposed approach is available in the Github repository *https://github.com/Afsana2910/iot-device-emulation*

Appendix II - List of Tables

1 Parameters for MQTT Client Information Configuration File 30

2 Parameters for Device Configuration File 30

3 Cron Job Parameters 40

4 University of Tartu Delta Centre Electricity Usage Dataset 44

5 University of Tartu Delta Centre Solar Level Dataset 51

Appendix III - List of Figures

1 Plot of Electricity Usage Level Data During a Working Day 21

2 Schema Fetched from the Input Dataset 22

3 Comparison of Original and Synthetic Electricity Consumption Data Generated by IoT data simulator 23

4 Electricity Usage Level During 8 hours of a Day in Original Dataset . . 24

5 Comparison of Synthetic Electricity Usage Level Generated using CT-GAN with Original Data 25

6 Comparison of Original and Synthetic Electricity Consumption Data Generated by Synsys 27

7 Key Software Components of Proposed Solution 36

8 Architectural Workflow of Proposed Solution 37

9 Illustration of Proposed Solution 38

10 Containerized Deployment of the Data Generator 42

11 Plot of Electricity Usage Level Data for a Month 45

12 Plot of Electricity Usage Level Data for a Week 45

13 Plot of Electricity Usage Level Data During a Working Day 46

14 Plot of Electricity Usage Level Data During a Weekend 46

15 Plot of Synthetically Generated Electricity Usage Level Data for a Week 48

16 Plot of Synthetically Generated Electricity Usage Level Data During a Working Day 49

17 Plot of Synthetically Generated Electricity Usage Level Data During a Weekend 49

18 Comparison Between Original and Synthetically Generated Electricity Consumption Data 50

19 Plot of Luminosity Level Data for a Week 51

20 Plot of Luminosity Level Data for a Typical Day 52

21 Plot of Synthetically Generated Luminosity Level Data for a Week . . . 54

22 Plot of Synthetically Generated Luminosity Level Data for a Day 54

Appendix IV - Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Afsana Khan**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Synthetic Sensor Data Generation for Authentic IoT Device Emulation,

   supervised by Pelle Jakovits and Mainak Adhikari.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Afsana Khan
04/08/2021