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 large-scale 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 scalable and reusable. In addition, the proposed approach is validated to measure the behavioral similarity of the synthetic IoT data with the real IoT traces.

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

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

Sünteetiline IoT-andmete genereerimine IoT-seadmete autentseks emuleerimiseks

Lühikokkuvõte: Asjade internet (IoT) on igapäevaelu füüsiliste objektide ühendamine üle interneti, et võimaldada neil üksteisega identifitseerimine ja suhtlemine. 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. Suuremate 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. Hindamisel kaalutakse nende lahenduste plusse ja miinuseid ning lisaks pakutakse välja lähenemine, mis toimiks tõhusamalt vajalikue nõuete täitmiseks, oleks skaleeritav ja korduvkasutatav. Lisaks valideeritakse välja pakutud andmete genereerimise lahendus, et mõõta sünteetiliste IoT-andmete käitumise sarnasus tõeliste IoT andmetega võrreldes.

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 6

1.2 Contribution to Thesis 8

1.3 Organization of Thesis 8

2 Literature Review **10**

2.1 Synthetic data generation 10

2.2 Existing Data Generation Tools and Simulators 11

2.3 CIISIM project 12

3 Experimental Analysis of Existing Data Generation Tools **14**

3.1 IoT Data Simulator 14

3.2 CTGAN 17

3.3 Mandrova: Sensor Data Generator for Python 19

3.4 log-synth 22

3.5 Iosynth 24

4 Proposed Approach for Synthetic IoT Data Generation **29**

5 Evaluation of the Proposed Approach **34**

6 Conclusion **43**

References **47**

Appendices **48**

Repository for Source Codes 48

List of Tables 48

List of Figures 48

Licence 50

1 Introduction

The Internet of Things (IoT) is a recent paradigm that refers to the network of a plethora of digital devices, software, and other technologies for connecting and exchanging data with other devices and systems through the internet. In other words, it is a system with interconnected devices provided with unique identifiers which can transfer data over a network without requiring human intervention. 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. However, concerns about the availability and security of privately owned data can limit research on IoT data.

1.1 Motivation

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, large-scale IoT systems have been upsurging. The complexity of these IoT systems is increasing exponentially as they grow[2]. The nature of the large-scale IoT-based

systems implies many challenges in terms of operation, security, quality control, and data management. 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. Besides, the issue of security and cost also exists. 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. 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. Hence, an effective solution to the problem needs to be brought forward.

The following research questions need to be addressed while designing an effective solution to generate synthetic data: 1. How to generate large-scale 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? and 3. How to quantify how close to the real data the generated synthetic data is?

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 stream 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 and its applications. Furthermore, it emphasizes the need to generate synthetic data in order to test IoT systems, as well as the challenges that may be encountered. In addition, state-of-the-art approaches to synthetic data generation will be described, and an overview of existing data generators is provided. Chapter 3 conducts an experimental analysis of available data generators by creating synthetic data for various use cases and weighing their

benefits and drawbacks. Later in chapter 4, a solution for synthetic IoT data generation is proposed and tested on specific use cases. The synthetic data generated from the proposed solution is then analyzed to determine its behavioral resemblance to real IoT traces. Finally, chapter 6 concludes the overall work by discussing potential applications and future extensions.

2 Literature Review

This chapter provides necessary information on the background of the thesis and the related works conducted so far in the field of synthetic data generation. The first section explores the methodologies for the generation of synthetic data. In the second section, an overview of the current data generators that have been developed and are in use has been provided. Lastly, the third section describes the project, which this thesis is a part of.

2.1 Synthetic data generation

The ideal approach would be to use existing real data instead of generating synthetic data. However, while there are numerous methods for acquiring datasets for development or research, there are many situations in which, for whatever reason, using real data is insufficient. For example, in the field of machine learning, well-known datasets such as the UCI (University of California, Irvine) machine learning repository have been widely used as benchmark datasets to test the performance of data analysis algorithms[3]. However, due to a large number of heterogeneous formats and distributions, it is difficult to use them in algorithm testing[4]. The privacy and confidentiality of real-world data is a common issue in database testing. Data masking is one solution to this problem[5]: the real data is replaced with generated data that has a reasonable level of similarity, which is commonly referred to as semi-synthetic data. IoT computing also has the need for data generation. Much of the relevant data is proprietary or subject to privacy restrictions, and it cannot be transferred to locations outside of the company's domain without posing significant risks to the business. The problem with data sharing is the risk of sensitive or proprietary information being leaked.[6]. 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. A tool was developed in [8] 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. Kuang et al. [9] designed, implemented, and evaluated time-series traffic generators for IoT applications. In [10] an efficient simulation methodology, which is particularly suitable to test IoT systems with a large number of interconnected devices in Urban environments from an application-layer perspective, has been proposed I.t provides the capability to simulate large-scale systems with thousands of geographically distributed 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 characteristics 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. W. Anderson et al. [6] design a synthetic IoT data generation framework. The framework is capable of generating terabytes of structurally similar synthetic data from a highly complex and nested original data source.

2.2 Existing Data Generation Tools and Simulators

For testing purposes, several open-source data generations and simulation tools are being used to replicate the behavior of actual data traces. The $QualityTesting(QT)$ tool itecasaleqt 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. Han et al. in [11] 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 [12] 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. The IoT-data-simulator iteIBAGroup82:online, 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. CTGAN [13] 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. The tool $SynSys$ [14] 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. Another open-source tool $IoSynth$ [15] is IoT device/sensor simulator and synthetic data generator, which allows generating data based on a rough schema. Schema-driven generation supports the generation of addresses, dates, foreign key references, unique id numbers, random integers, sensor data, etc.

2.3 CIISIM project

This thesis is related to 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 that 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 on a larger scale and at a lower cost. Since the hardware components are virtualized, this means these components need to generate data behaving similarly to

the actual ones. As a result, a suitable synthetic data generation approach is required in order to achieve the project goals.
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.
- Can create multiple copies of IoT devices, each with a unique UUID, and generate data as required.
- Have the functionality to send the generated IoT data to MQTT broker
- Be able to automate the deployment of emulated devices.
- Can be deployed in a containerized manner (e.g. using Docker)

The goal of this thesis is to fill this gap, to 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.

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 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. The documentation of these solutions was studied and later evaluated by generating synthetic data similar to an open dataset. The experimental results are as follows:

3.1 IoT Data Simulator

The IoT-data-simulator [16] 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 Household Electric Power Consumption dataset [17] has been used as an input in order to generate similar behaving synthetic data. The Household Power Consumption dataset is a multivariate time series of a single household's electricity consumption over a four-year period. In addition to date and time,

the dataset comprises seven variables which are:

- **global_active_power:** The total active power consumed by the household (kilowatts).
- **global_reactive_power:**The total reactive power consumed by the household (kilowatts).
- **voltage:** Average voltage (volts).
- **global_intensity:** Average current intensity (amps).
- **sub_metering_1:** Active energy for kitchen (watt-hours of active energy).
- **sub_metering_2:** Active energy for laundry (watt-hours of active energy).
- **sub_metering_3:** Active energy for climate control systems (watt-hours of active energy).

	Date	Time	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_metering_3
0	16/12/2006	17:24:00	4.216	0.418	234.84	18.4	0.0	1.0	17.0
1	16/12/2006	17:25:00	5.360	0.436	233.63	23.0	0.0	1.0	16.0
2	16/12/2006	17:26:00	5.374	0.498	233.29	23.0	0.0	2.0	17.0
3	16/12/2006	17:27:00	5.388	0.502	233.74	23.0	0.0	1.0	17.0
4	16/12/2006	17:28:00	3.666	0.528	235.68	15.8	0.0	1.0	17.0
5	16/12/2006	17:29:00	3.520	0.522	235.02	15.0	0.0	2.0	17.0
6	16/12/2006	17:30:00	3.702	0.520	235.09	15.8	0.0	1.0	17.0
7	16/12/2006	17:31:00	3.700	0.520	235.22	15.8	0.0	1.0	17.0

Figure 1. Household Electric Power Consumption Dataset

The figure 1illustrates the Household Electricity Consumption Dataset's contents. The IoT data simulator takes this dataset as input and constructs a schema from it, as illustrated in Figure 3. By modifying the values, the simulator can generate very similar data 2 for these variables in the actual dataset. As can be seen from the generated synthetic data, the date property has been altered, but the time property has remained unchanged, as data can be generated here with the same time intervals as in the original

dataset. Additionally, the simulator supports sending generated data as a payload to target systems such as AMQP, Kafka, Local object storage, MQTT, and REST

	Date	Time	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_metering_3
0	2021-05-06	17:24:00	3.854470	0.403603	235.562368	18.762243	0	1	15
1	2021-05-06	17:25:00	4.438276	0.421450	234.894119	21.614812	0	1	16
2	2021-05-06	17:26:00	4.131116	0.378465	236.628292	22.940113	0	2	15
3	2021-05-06	17:27:00	4.240587	0.500315	232.368766	23.938674	0	1	19
4	2021-05-06	17:28:00	4.545904	0.496375	244.409744	23.467746	0	1	15
5	2021-05-06	17:29:00	5.833591	0.340601	247.642923	20.677840	0	2	17
6	2021-05-06	17:30:00	5.196084	0.569069	234.445633	21.450840	0	1	17
7	2021-05-06	17:31:00	3.978307	0.363023	235.729169	23.827369	0	1	16

Figure 2. Synthetic Data Generated Using IoT Data Simulator

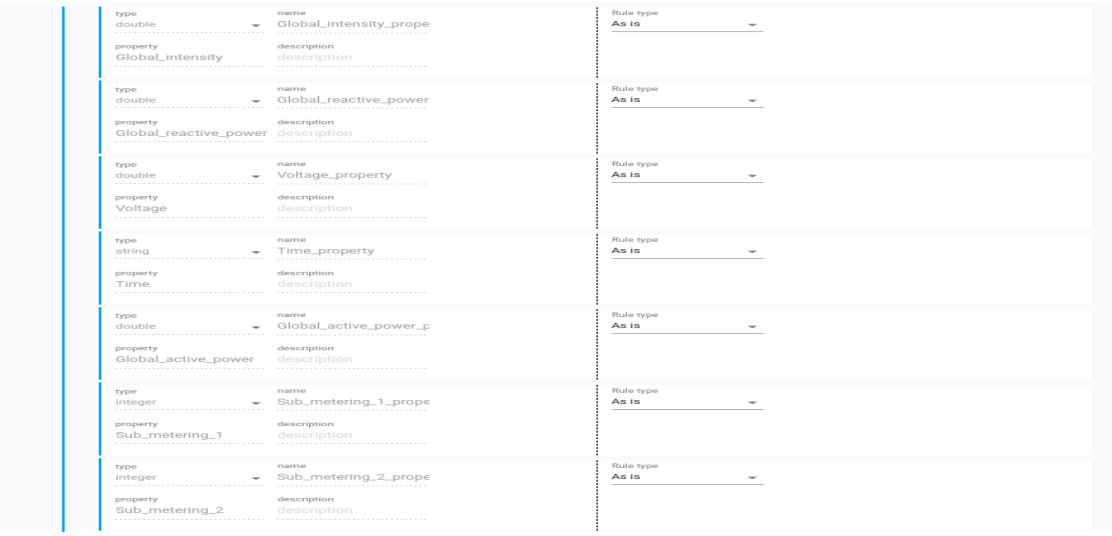

Figure 3. Schema Fetched from the Input Dataset

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 when comparing the original and synthetic data since the output did not show the data spikes and falls as in the actual dataset.

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 [13] 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, whether these are string values or not.

For trying out the CTGAN module, the Intel Lab Dataset [18] has been used as input. The dataset contains 8 columns: data, time, epoch, moteid, temperature, humidity, light, and voltage. Here epoch is a monotonically increasing sequence number from each mote. Moteids range from 1-54. Temperature is in degrees Celsius. Humidity is temperature corrected relative humidity, ranging from 0-100%. Light is in Lux (a value of 1 Lux corresponds to moonlight, 400 Lux to a bright office, and 100,000 Lux to full sunlight.) Voltage is expressed in volts, ranging from 2-3V. Therefore in order to match the input

format of CTGAN, the Intel Lab Dataset had to be preprocessed. The instances with missing values had to be removed, the type of the columns had to be changed to int, float or str. Besides the values in the columns which seemed anomalous (did not match the specifications of the dataset stated before) were removed. The original data looks like the table shown in Figure 4. The number of samples (synthetic data) to be generated can be customized. For this experiment, 500 samples have been generated5.

	date	time	epoch	moteid	temperature	humidity	light	voltage
0	2004-03-31	03:38:15	2	1.0	122.153	-3.919	11.04	2.034
1	2004-02-28	00:59:16	3	1.0	19.988	37.093	45.08	2.700
2	2004-02-28	01:03:16	11	1.0	19.302	38.463	45.08	2.687
3	2004-02-28	01:06:16	17	1.0	19.165	38.804	45.08	2.687
4	2004-02-28	01:06:46	18	1.0	19.175	38.838	45.08	2.700

Figure 4. Intel Lab Dataset

	date	time	epoch	moteid	temperature	humidity	light	voltage
355	2004-03-12	00:59:16	190	1.84	19.008	40.143	611	2.772
346	2004-02-28	00:59:16	24.9	7.66	19.493	40.560	87	2.685
405	2004-02-28	00:59:16	187	3.82	19.182	44.095	315	2.761
292	2004-02-28	01:00:46	94	1.25	19.967	39.407	95	2.642
237	2004-02-28	01:00:46	84.4	3.54	18.799	38.739	37	2.644

Figure 5. Synthetic Data Generated using CTGAN

It was observed that the synthetic data generated using CTGAN is similar to the original data only in cases where the attributes or columns have the type float. While in case of the columns having type integer, some anomalies were detected. In the experiment, the columns epoch and moteid were integers, while the columns temperature, humidity, light, and voltage were float numbers. After the generation of synthetic data, it

was observed that the values generated for epoch and moteid did not seem normal and were different from the original data. Although they were integer values, the values were found to be float numbers after generating synthetic data. Also, there were below zero values, but having negative values for epoch and moteid does not seem logical. However, the values generated for the columns temperature, humidity, light, and voltage were okay. In order to compare the distribution of data between the original dataset and the synthetic data, histograms were plotted (Figure 6)for the columns temperature, humidity, light, and voltage.

3.3 Mandrova: Sensor Data Generator for Python

Mandrova [19] is a free and open-source Python application for the generation of scientific data. This application makes it simple to produce data from any distribution. The user must specify the distribution and its associated parameters in order to generate data. It may be advantageous in situations where IoT data must be produced with a particular distribution. However, only the most commonly used distributions, such as the normal, exponential, and gamma distributions, are implemented in the application.

Initially, the application is experimented by generating data for a temperature sensor following a normal distribution. The mean mu and standard deviation $sigma$ is set to 0 and 1 respectively. 100 samples of data were generated which is illustrated in Figure:7.

```
from data_generator import SensorDataGenerator as sdg
dg = sdg()
dg.generation_input.add_option(sensor_names="temperature_sensor",
                               distribution="normal",
                               mu=0,
                               sigma=1)
dg.generate(sample_size=100)
```

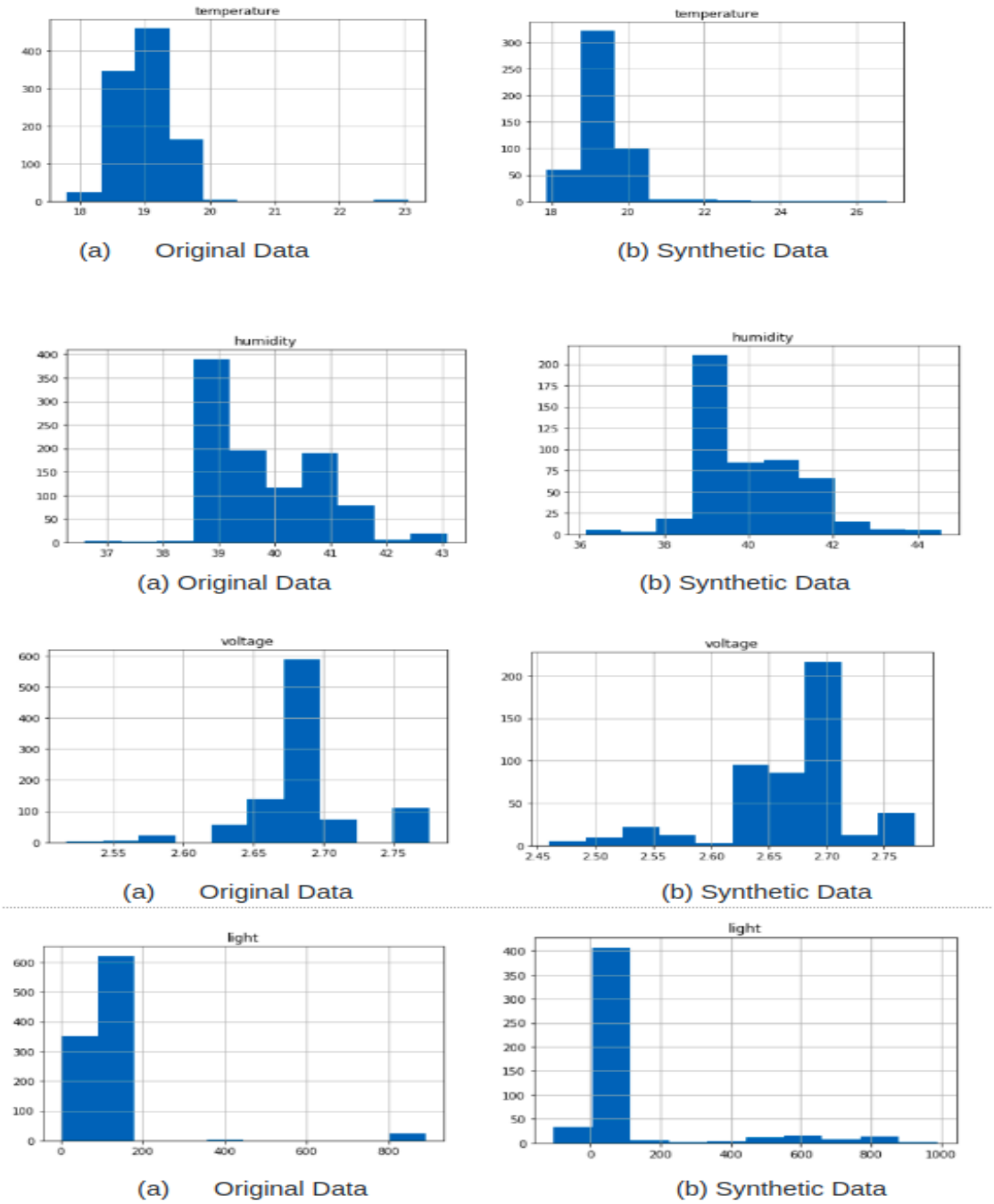

Figure 6. Histogram Comparing Distributions of Original and Synthetic Data Generated using CTGAN

Additionally, the other distributions were also tested by generating data for multiple sensors with different distributions. The application is capable of generating data for a large number of sensors with varying distributions.

```
dg.generation_input.add_option(sensor_names="temperature_sensor",
    distribution="normal", mu=0, sigma=1)
dg.generation_input.add_option(sensor_names="humidity_sensor",
    distribution="gamma", alpha=1.5, beta=1.8)
dg.generation_input.add_option(sensor_names="solar_sensor",
    distribution="exponential", lambd=3.5)
```

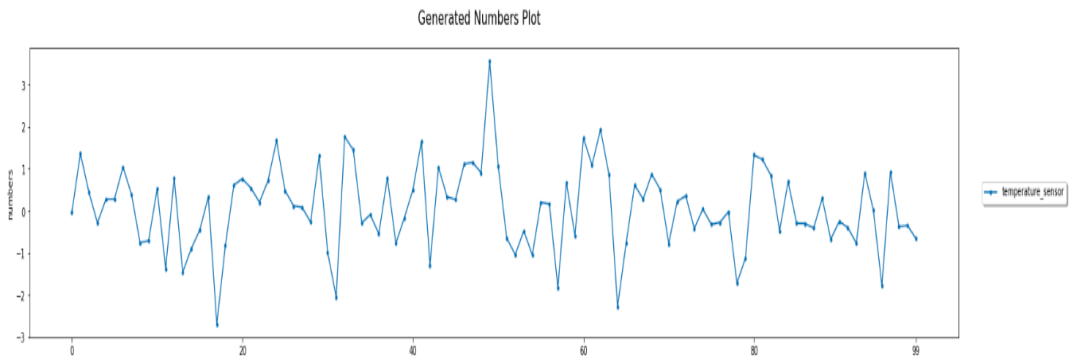

Figure 7. Generation of Data with Normal Distribution for Single Sensor

The application is also able to generate data based on functions. Figure 9 illustrates the generation of data using the function $xsin(x)$ where x is initialized to 0 and $step$ is set to 0.03. In general, Mandrova appears to be an interesting application for generating IoT data when actual traces are not available but the user knows what distribution data should be produced. However, the data produced here is quite random and does not bear any behavioral similarity to data generated by real-world IoT devices. Additionally, this application's configuration choices are limited to particular distributions and are difficult to configure to meet user requirements.

Figure 8. Generation of Data with Varying Distributions for Multiple Sensors

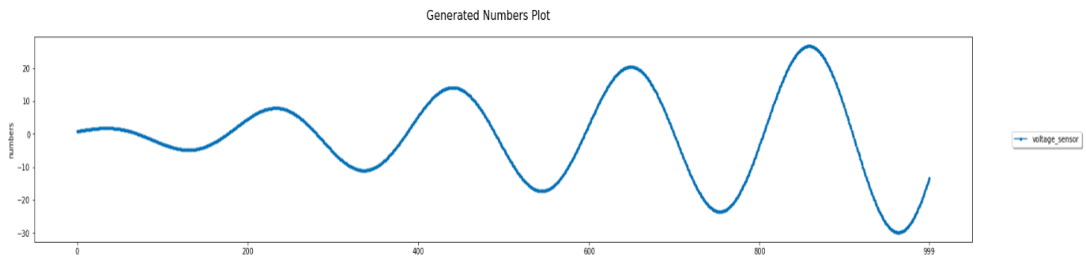

Figure 9. Generation of Data by Specifying a Function

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.

Noise(dB)	
mean	71.586957
std	0.658030
min	70.70000
max	73.70000

Table 1. Statistics of Noise Measurement in Original Dataset

Noise(dB)	
mean	71.402811
std	0.894442
min	70.146803
max	73.816769

Table 2. Statistics of Noise Measurements Generated Synthetically Using log-synth

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]
```

For experimenting with log-synth, the road traffic noise dataset of Thessaloniki, Greece has been used. The dataset basically contains the traffic road noise every hour and every day from 2004 to 2008. Each row of the dataset provides three pieces of information; the date when the reading was taken, time of measurement, and measured values in decibels(dB). Analyzing the dataset, it was observed that the road noise measurements were random values having a certain range and did not quite have a pattern in general. An idea of the measurements for a single day could be achieved from the statistics as shown in Table 1. It can be observed from Table 1, the range of measurements varies from 71 to 73 decibels. This information is specified in the schema, and the distribution is selected as $random_walk$.

```
[
    {
  "class":"sequence",
  "length": 3,
  "base":{
      "class":"map",
      "value": [
    {
```

```
            "name": "sample-time",
            "class": "event",
            "format":"yyyy-MM-dd HH:mm:ss",
            "start": "2021-08-05 1:00:00",
                        "rate": "1/h"
        },
        {
            "name": "temperature",
            "class": "random-walk",
            "mean":71.586957,
            "sd":0.658030,
            "start":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 road noise measurements. As shown in Table 2, the produced synthetic data has statistical values that are comparable to those found in the original dataset.

Though log-synth works well for generating random data with defined distributions and ranges, 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, rendering it unusable.

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 fre-

quency 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. Table 3describes 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 3. 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 4. Parameters for Device Configuration File

The device configuration file is essentially a JSON template that specifies the parameters (Table 4) 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.

We experimented with the Iosynth simulator using the Thessaloniki, Greece, road traffic noise 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. 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":"noise", "min":70.7, "max":73.7}
        ]
    }
]
```

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 road noise dataset, the noise intensity value is created using the $double_walk$ generator, which generates random noise levels within a defined range determined by the dataset's analysis previously. As demonstrated in Tables 5 and 6, 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. This feature of Iosynth is a part of what we wanted to achieve while generating synthetic data. 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.

Noise(dB)	
mean	71.586957
std	0.658030
min	70.70000
max	73.70000

Table 5. Statistics of Noise Measurement in Original Dataset

Noise(dB)	
mean	71.52318
std	0.602417
min	70.54912
max	73.97521

Table 6. Statistics of Noise Measurements Generated Synthetically Using Iosynth

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 specific IoT device behaviors. 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. Replicating this form of behavior was the main challenge to overcome.

It was obvious that an effective solution was needed to meet the previously stated requirements. Initially, it was not clear whether a completely new solution was needed or existing ones could be adapted. So, later we decided to investigate which of the data generators previously experimented with fulfilled the requirements of the CIISIM project stated in section 2.3 to a wider extent. 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.

As described in the previous chapter, 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. To emulate the behavior of real IoT devices based on other factors such as the time of day, month, or season the Iosynth can be adapted to use different device configuration files having varying configurations. Also, the execution of Iosynth could be scheduled based on other factors to achieve the required results.

In this thesis, we have proposed a solution where Iosynth is adapted to use multiple device configuration files rather than a single one, generating data with varying configurations depending on other conditions. The simulator Iosynth is launched using a single Java command that includes the MQTT client details and configuration file as parameters. To accomplish our goal, the Java command is run at predefined intervals, each time varying the device configuration file and passing it as the parameter. For this purpose, Iosynth is scheduled to run in this manner using the software utility $Cron$, which is also known as $cronjob$. $Cron$ 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
- The output or action of the script that is determined by the function of the script being invoked.

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 7. Cron Job Parameters

The Cron job is specified in the following format shown below where the first 5 parameters are described in Table 7.

```
* * * * * [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. To address this problem, rather than directly running the Java command, a Python script has been used to invoke the Iosynth simulator by running the Java command with different device configuration files only after all previous Java processes involving running Iosynth have been terminated. Additionally, the Python script handles the logic associated with the selection of device configuration files to be passed as a parameter to the Java command.

Moreover, The distributions like uniform, normal, exponential, random-walk are allowed by Iosynth are implemented as separate Java classes. To make a custom solution, and to mimic the pattern behavior described with the example of luminosity earlier,

we also needed linearly increasing and decreasing generators. Hence these functions were implemented as Java classes (*SensorDoubleIncrement.java* & *SensorDoubleDecrement.java*) as a modification to the Iosynth simulator.

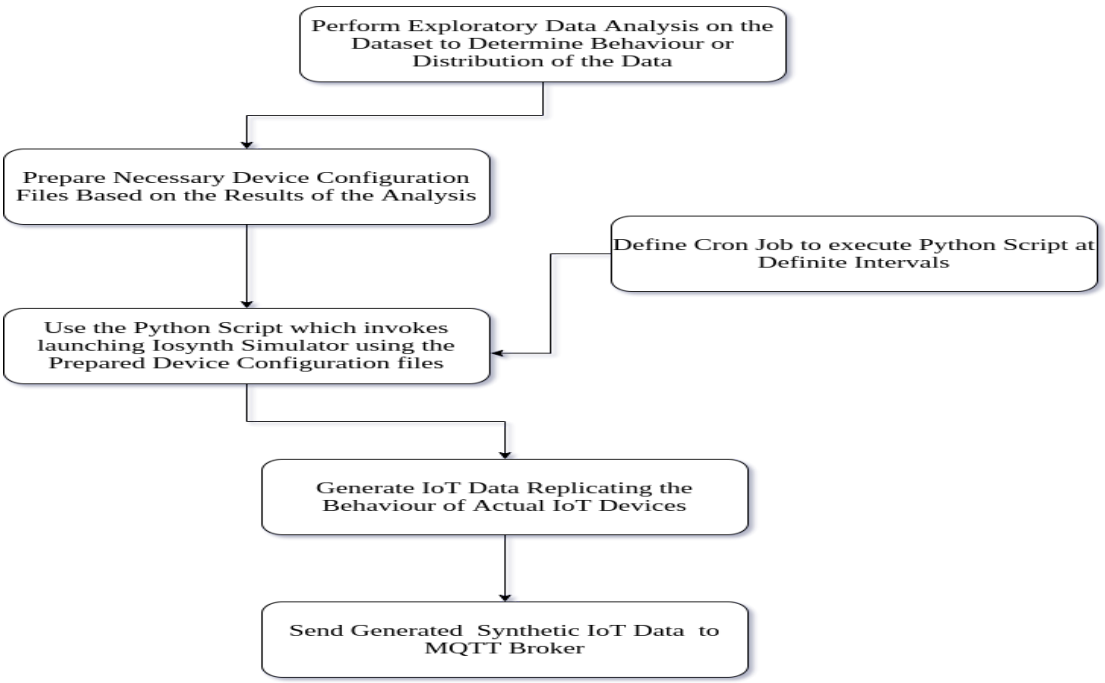

Figure 10. Architecture of Proposed Solution

Figure 11 depicts the architecture of the proposed solution. The overall summary for synthetic data generation 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. This assists in determining whether the data follows a specific distribution or exhibits random behavior. Additionally, statistical details of the data may be obtained, such as the data's minimum or maximum range.

2. The device configuration files are created based on the analysis from the previous

step. It holds information such as the frequency in which data should be produced, the number of devices for which data should be generated, and the distribution of the data, including the minimum and maximum range of the data.

3. The Python script invokes the Iosynth simulator by using the prepared device configuration files and passing them as a parameter to the Java command. Additionally, the script handles the logic for selecting which device configuration file to use.

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 the second step.

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

5 Evaluation of the Proposed Approach

In order to evaluate the quality of the proposed approach, the University of Tartu Delta Centre Solar Level Dataset has been used where the objective is to generate similar behaving IoT data as that in the dataset. This dataset contains data about the outside luminosity level from $2020/09/2$ to $2020/11/23$ collected from the sensors installed at the Delta Centre. The contents of the dataset are described in Table 8.

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 8. University of Tartu Delta Centre Solar Level Dataset

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 behavioral pattern of these data was determined by observation from the graphs generated.

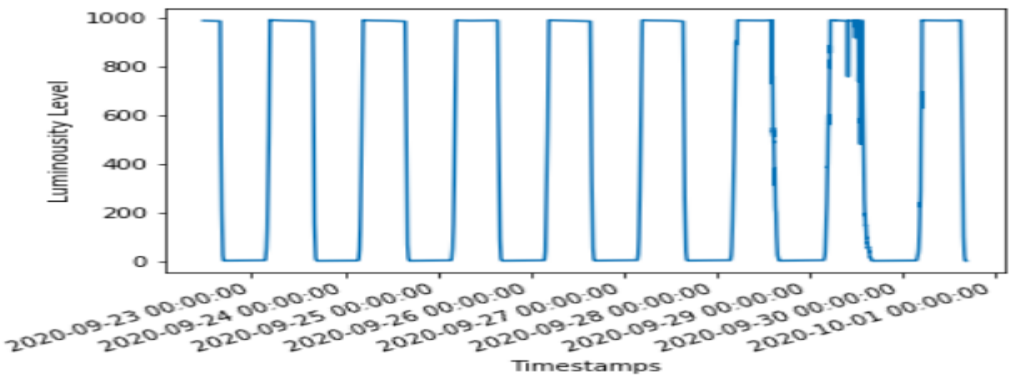

Figure 11. Plot of Luminosity Level Data for a Month

Figure 12. Plot of Luminosity Level Data for a Day

From the figures 11 and 12, one can get an idea about the behavioral pattern of the luminosity level in a usual day. According to the graph is figure 12 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 4:00), 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 month. The trends such as the rise and fall of the luminosity levels were identified. Four trends were determined which were: the stable state of the luminosity level before 4 am, the rising state of luminosity level after 4 am, the stable state of peak luminosity level during mid-day, and finally the declining state of luminosity level after 4 pm. Hence, four device configuration files were configured to represent the trends of luminosity level. The python script that invokes the Iosynth simulator makes use of these device configuration files to generate synthetic luminosity data with a pattern close to that of the actual dataset.

Maximum Luminosity Level Observed	998.60 LUX
Average Luminosity Level Observed	582.38 LUX
Minimum Luminosity Level Observed	0.60 LUX

Table 9. Statistical Metrics in University of Tartu Solar Level Dataset

```
{
        "type":"simple",
        "uuid":"dev%04d",
        "topic":"device/{$uuid}",
        "sampling":{"type":"fixed", "interval":300000},
        "copy":1,
        "sensors":[
            {"type":"dev.timestamp",     "name":"ts"},
            {"type":"dev.uuid",          "name":"uuid"},
            {"type":"double_uniform", "name":"luminousity", "min":0,
                "max":1}

        ]
}
```

Listing 1. Configuration for Generating Luminosity Level During Evening and Night

```
{
        "type":"simple",
        "uuid":"dev%04d",
        "topic":"device/{$uuid}",
        "sampling":{"type":"fixed", "interval":300000},
        "copy":1,
        "sensors":[
            {"type":"dev.timestamp",     "name":"ts"},
            {"type":"dev.uuid",          "name":"uuid"},
            {"type":"double_increment", "name":"luminousity", "min
                ":0.0, "max":990.0}

        ]
}
```

Listing 2. Configuration for Generating Luminosity Level During Early Morning

```
{
        "type":"simple",
        "uuid":"dev%04d",
        "topic":"device/{$uuid}",
        "sampling":{"type":"fixed", "interval":300000},
        "copy":1,
        "sensors":[
            {"type":"dev.timestamp",     "name":"ts"},
            {"type":"dev.uuid",          "name":"uuid"},
            {"type":"double_uniform", "name":"luminousity", "min
                ":990, "max":1000}

        ]
}
```

Listing 3. Configuration for Generating Luminosity Level During Mid Day

```
{
        "type":"simple",
        "uuid":"dev%04d",
        "topic":"device/{$uuid}",
        "sampling":{"type":"fixed", "interval":300000},
        "copy":1,
        "sensors":[
            {"type":"dev.timestamp",     "name":"ts"},
            {"type":"dev.uuid",          "name":"uuid"},
            {"type":"double_decrement", "name":"luminousity", "min
                ":0.0, "max":990.0}

        ]
}
```

Listing 4. Configuration for Generating Luminosity Level During Afternoon

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. A Cron job was scheduled to execute the python script every hour which invoked the Iosynth simulator using the device configuration files.

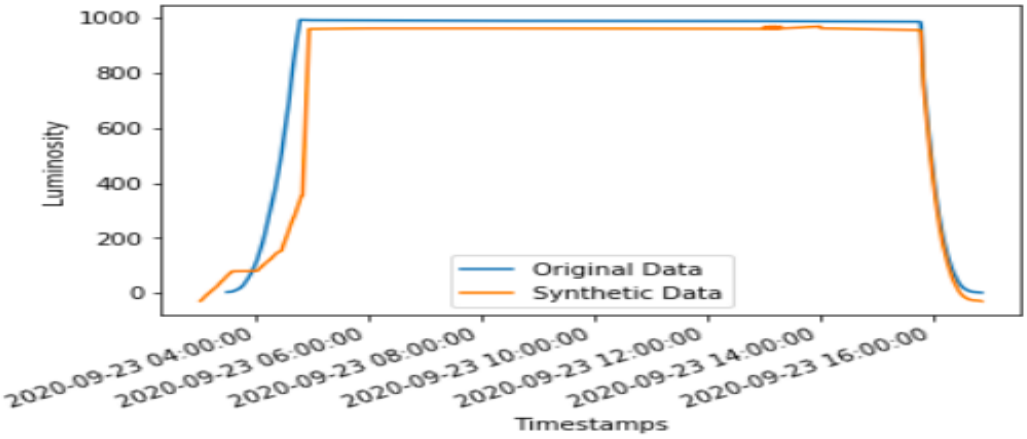

Figure 13. Plot of Synthetically Generated Luminosity Level Data for a Day

Maximum Luminosity Level Observed	990.00 LUX
Average Luminosity Level Observed	582.78 LUX
Minimum Luminosity Level Observed	0.42 LUX

Table 10. Statistical Metrics in Synthetically Generated Dataset

One of the most common methods for assessing the quality of synthetically generated data is to conduct exploratory data analysis on it[20]. This includes comparing

statistical measures to the original dataset and visual inference, which involves plotting synthetic and original data to determine how similar they are. The figure 13 illustrates that the luminosity dataset which has been generated synthetically exhibits a similar pattern to the original dataset, with data spikes and falls occurring at the same timestamps. Apart from that, the statistical metrics such as maximum, average, and minimum values for the synthetic luminosity data are reasonably similar to those for the real data. However, it is not always possible to visually justify the similarity between the synthetic and original data. In such cases evaluating the distribution of data in both cases could be useful. Data Distribution is a function that returns a list of all the possible values for the Data. It can be either continuous or discrete. Numerous well-known standard Probability Distribution functions estimate the likelihood of various possible outcomes occurring in an experiment. Several well-known probability distributions have a standard form, including the Normal, Log-Normal, Beta, and Gamma. However, the difficulty is that real-world data may not always follow the well-known Probability Distributions. In this case, we can approximate the most likely Probability Distribution and validate its goodness of fit[21].

The Chi-square test [22] can be used to determine the goodness of fit. It is a statistical test that determines how well a model fits actual observed data. It is useful for determining whether two variables are related or independent of one another, as well as for determining the goodness-of-fit between an observable distribution and a theoretical distribution of frequencies. For experimentation, the Chi-square test is used for both Original data and the Synthetically generated data in order to approximate their data distributions. Exhibiting the same data distribution indicates the fact that, the synthetic data has been generated close enough to the actual data. From the tables 11 and 12, it is observed that the top three probability distributions in the case of original data are Beta, Uniform, and Inverse Gaussian. A similar case is found in the case of synthetic data as well. Therefore, it can be said that both the original data and synthetic data approximately follow Beta distribution which indicates the similarity in the synthetic data generated with the original data.

Another approach to evaluate the similarity of synthetic data can be the Kolmogorov–Smirnov One-Sample Test[23]. This test is used in situations where a comparison has to be made between an observed sample distribution and theoretical distribution. The test is based on hypothesis testing where two hypotheses are considered. The null hypothesis H_0 and the alternate hypothesis H_1.

Distribution	**chi-square statistic**
beta	0.283702
uniform	0.961730
invgauss	90.638302
pearson3	167.744283
lognorm	177.519735
gamma	196.319266
norm	200.807041
triang	268.408946
expon	530.872901
weibull_max	5945.961796
weibull_min	14527.325975

Table 11. Distributions Listed By Goodness of Fit for Original Data

Distribution	chi-square statistic
beta	6.509914
uniform	17.105017
invgauss	103.561738
lognorm	143.012687
gamma	164.045659
norm	168.491857
triang	473.399976
expon	603.750282
weibull_max	5616.930060
weibull_min	13954.213087
pearson3	28590.355224

Table 12. Distributions Listed By Goodness of Fit for Synthetic Data

The hypothesis considered for the testing are as follows:

- H_0**:**The original and synthetic data follow the same data distribution
- H_1**:**The original and synthetic data do not follow the same data distribution

The method to perform this testing is to first determine the K-S statistic and p-value followed by comparing it with the significance level (standard: 5%). In this case, the statistic has been calculated comparing the original and statistic using python $scipy.stats$ as follows:

```
from scipy.stats import ks_2samp
import numpy as np
ks_2samp(original_data, synthetic_data)
```

If the K-S statistic is small or the p-value is high (greater than the significance level, 5%), then the null hypothesis cannot be rejected. Conversely, we can reject the null hypothesis

if the p-value is low. The calculated K-S statics for our case was 0.0308 which is less than the critical value at the 5% significance level obtained from the K-S Table. The p-value was obtained to be 0.47 which is higher than the significance of 5%. Thus, we fail to reject the null hypothesis H_0 which states that the original and synthetic data follow the same data distribution. Hence it can be concluded that in the experiment conducted the synthetically generated luminosity data is close to the original data of the dataset.

6 Conclusion

This thesis has focused on the challenges involved in synthetic IoT data generation required for testing large-scale IoT systems. The objectives of the thesis included generating large-scale IoT data with customizable frequency without physically deploying IoT devices, generating synthetic data to replicate the real data generated by IoT devices, taking seasonality and other patterns into account, and evaluating how similar the generated synthetic data was to the real data. In summary, the following contributions were made to accomplish these objectives:

- Different data generation tools were studied and analyzed to evaluate how they performed. Experiments were conducted by replicating existing open-source IoT datasets. This was done to decide if there was a necessity of developing a new data generator or the modification of the existing solution was enough.
- We chose a suitable data generator which was later modified by implementing extra distributions for data generation.
- An approach has been proposed using the modified data generator and scheduling it using $Cron$ which could emulate behavioral patterns of real IoT devices such data spikes and falls during certain time periods.
- We tested the proposed approach by experimenting with it with a dataset. Finally, the results obtained were evaluated to check the similarity with the original dataset by visual inference and other statistical measures.

However, the proposed approach has some limitations that can be considered as the future scope of the work. The future directions of the proposed approach are highlighted as follows:

- The proposed approach involves analysis of the dataset and expects the user to prepare device configuration files. Automation of this step would be more efficient

where the script used for the analysis of the dataset generates device configuration files automatically based on the trends and patterns of the data in the actual dataset.

- The data generator chosen can be modified to some extent additionally to perform tests for longer periods of time without the need to actually wait that long.
- In addition, the proposed approach can be extended to adapt the generation of various forms of IoT data, including GPS coordinates, data based on actuators, and so on. This is a requirement for the CIISIM project's Smart Scooter use case, which will be worked on in the near future.

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. We will be using this approach in the project CIISIM (section 2.3) which involves creating a Virtual village comprising of virtualized IoT devices. The virtual IoT devices will be set up as docker containers and the extension of the data generator will be 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] Arthur Asuncion. Uci machine learning repository, university of california, irvine, school of information and computer sciences. *http://www. ics. uci. edu/˜ mlearn/ML-Repository. html*, 2007.

[4] R Syahaneim, A Hazwani, N Wahida, SI Shafikah, and PN Ellyza. Automatic artificial data generator: Framework and implementation. In *2016 International Conference on Information and Communication Technology (ICICTM)*, pages 56–60, 2016.

[5] N Ltd. Data masking: What you need to know. *A Net 2000 Ltd. White Paper*, 25, 2000.

[6] 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.

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

[9] Jilong Kuang, Daniel G Waddington, and Changhui Lin. Techniques for fast and scalable time series traffic generation. In *2015 IEEE International Conference on Big Data (Big Data)*, pages 105–114. IEEE, 2015.

[10] Giacomo Brambilla, Marco Picone, Simone Cirani, Michele Amoretti, and Francesco Zanichelli. A simulation platform for large-scale internet of things scenarios in urban environments. In *Proceedings of the First International Conference on IoT in Urban Space*, pages 50–55, 2014.

[11] 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.

[12] 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.

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

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

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

[16] 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).

[17] Uci machine learning repository: Individual household electric power consumption data set. https://archive.ics.uci.edu/ml/datasets/individual+household+electric+power+consumption. (Accessed on 05/07/2021).

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

[19] makinarocks/mandrova: An awesome synthetic sensor data generator for python3. https://github.com/makinarocks/Mandrova. (Accessed on 05/14/2021).

[20] Andreas Buja, Dianne Cook, Heike Hofmann, Michael Lawrence, Eun-Kyung Lee, Deborah F Swayne, and Hadley Wickham. Statistical inference for exploratory data analysis and model diagnostics. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 367(1906):4361–4383, 2009.

[21] RalphB D'Agostino. *Goodness-of-fit-techniques*. Routledge, 2017.

[22] William G Cochran. The $\chi 2$ test of goodness of fit. *The Annals of mathematical statistics*, pages 315–345, 1952.

[23] Frank J Massey Jr. The kolmogorov-smirnov test for goodness of fit. *Journal of the American statistical Association*, 46(253):68–78, 1951.

Appendices

Appendix I - Repository for Source Codes

The source codes for the extended version of Iosynth simulator is located in the Github repository *https://github.com/Afsana2910/iosynth*.

Appendix II - List of Tables

1 Statistics of Noise Measurement in Original Dataset 23

2 Statistics of Noise Measurements Generated Synthetically Using log-synth 23

3 Parameters for MQTT Client Information Configuration File 25

4 Parameters for Device Configuration File 26

5 Statistics of Noise Measurement in Original Dataset 28

6 Statistics of Noise Measurements Generated Synthetically Using Iosynth 28

7 Cron Job Parameters 31

8 University of Tartu Delta Centre Solar Level Dataset 34

9 Statistical Metrics in University of Tartu Solar Level Dataset 36

10 Statistical Metrics in Synthetically Generated Dataset 38

11 Distributions Listed By Goodness of Fit for Original Data 40

12 Distributions Listed By Goodness of Fit for Synthetic Data 41

Appendix III - List of Figures

1 Household Electric Power Consumption Dataset 15

2 Synthetic Data Generated Using IoT Data Simulator 16

3 Schema Fetched from the Input Dataset 16

4 Intel Lab Dataset 18

5 Synthetic Data Generated using CTGAN 18

6 Histogram Comparing Distributions of Original and Synthetic Data Generated using CTGAN 20

7 Generation of Data with Normal Distribution for Single Sensor 21

8 Generation of Data with Varying Distributions for Multiple Sensors . . 22

9 Generation of Data by Specifying a Function 22

10 Architecture of Proposed Solution 32

11 Plot of Luminosity Level Data for a Month 34

12 Plot of Luminosity Level Data for a Day 35

13 Plot of Synthetically Generated Luminosity Level Data for a Day 38

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
14/05/2021