VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Rihards Kellers

Home Automation Network MODBUS

IOT-Ticket-MODBUS Network Software and Implementation

Technology and Communication
2021

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

ABSTRACT

Author	Rihards Kellers
Title	Home Automation Network MODBUS
Year	2021
Language	English
Pages	36
Name of Supervisor	Jani Ahvonen

There are many IoT networks today, which include mainly wireless devices. While that might sound convenient in many situations, wireless IoT devices might be vulnerable to hackers. This thesis aims to implement a robust IoT network using RS-485 electronic signaling to transmit data using a Modbus protocol known from the industry. Finally, the master (PC) transfers the data to the IoT ticket cloud service.

The aim of this thesis was to create a Modbus protocol-based IoT network. A personal computer with USB to RS-485 adapter includes a program written using the Python programming language as a MODBUS master and data processor for such network implementation. Arduino single-board microcontrollers were programmed to be MODBUS slave devices.

Keywords Internet of things, MODBUS, Arduino, Python

CONTENTS

LIST OF FIGURES AND TABLES

1 INTRODUCTION

IoT networks are becoming more popular as technology advances. Interconnected devices within a home, office or industrial buildings are pretty common to monitor and control the environment. Today it is easy to get a voice assistant, connect to the house ethernet network, control lights, and sound in the household.

Products on the market are widespread and famous. However, consumers might be concerned about the privacy and security of a household and private life when all devices are in the same network. IoT devices can be vulnerable to hackers as users can be careless or bugs exist within a specific device that allows unauthorized access to personal data, such as voice command history.

The Modbus RTU protocol is an excellent alternative to wireless ethernet-connected IoT devices. The RS-485 serial connection-based the Modbus RTU network consists of a master device and up to 247 custom devices that support Modbus RTU or Modbus RTU certified devices over long distances.[1]

The Modbus protocol on its own is the only tool for interaction between the devices. As the Modbus protocol is a basic machine-to-machine protocol, separate software must process both the Modbus network and interact with the IoT-Ticket cloud.
In this setup, a personal computer with an Ubuntu operating system serves as a master device in the Modbus network. USB to RS-485 device was used to create

[1] Full guide to serial communication protocol and our RS-485

software that would be a layer between the Modbus network and data storage, in this case, IoT-TICKET chosen as an IoT data storage and control platform.
Software that allows Modbus devices and IoT-TICKET cloud to work as the same system was programmed using the Python programming language due to its high versatility and many libraries.
However, as an operating system, such as Microsoft Windows also supports Python, it is possible to use the home server as a Modbus master device or turn Raspberry Pi into a Modbus master device.

The system designed within this thesis can also be appliable to automation and device design studies in general. Being open-sourced protocol, this system can allow students to design, develop and implement their own devices, interacting with the system, allowing them to apply the theoretical knowledge in practice.

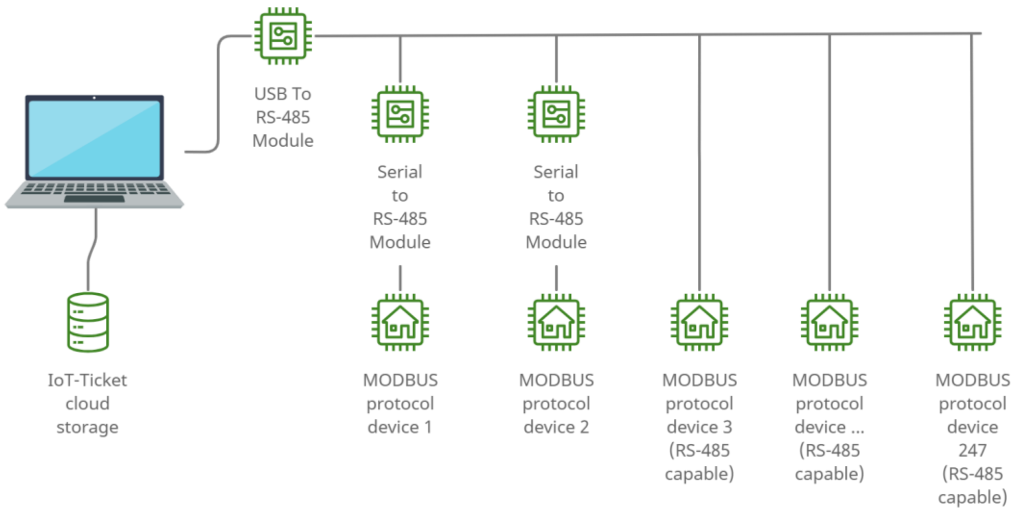


Figure 1. Network topology of IoT-Ticket-Modbus network with one computer and five slave devices.

2 MODBUS PROTOCOL

2.1 About Modbus Protocol

The Modbus protocol is the most commonly used industrial protocol designed for Machine-to-Machine interaction; developed by Modicon company, later acquired by Schneider Electric, the protocol was introduced in the market in 1979. Schneider Electric assisted in the development of independent developer and user community organization.[2]

The open-source base and versatility of the protocol allow multiple manufacturers to adopt the protocol and implement it in their production. The main applications of the Modbus protocol are relay and controller handling, sensor data collection, and monitoring.

2.2 MODBUS RTU Protocol

The system implemented in this thesis focuses on the MODBUS RTU frame format on the RS-485 half-duplex serial data line.

The Modbus RTU frame format encodes data into binary format; a time interval performs the role of a package divider. This protocol has a low tolerance towards latency.

Modbus devices interact following the master-slave model. All requests are made only by the master device; slave devices can only respond to the requests and cannot independently begin data transfer.

The Modbus network can consist of multiple segments; however, one segment can include only one master device and 247 slave devices. Each slave device has a unique address within the network in a range from 1 to 247. The master device

[2] Modbus FAQ: About The Modbus Organization.

does not have an address. However, the master device can send a package addressing address 0, which will broadcast a package to all slave devices; slave devices cannot respond to broadcast requests. Broadcasting will not be implemented within the scope of this thesis since there is no confirmation response from the slave devices.

2.3 MODBUS RTU Package Application Data Unit Structure

MODBUS RTU Application Data Unit consists of:

1. Slave device address – Address of the package recipient device in range from 0 to 247. (0 for broadcasts)
2. Protocol data unit — Main part of the package, which includes the function code and data. Data depends on the function code — the maximum size of 253 bytes.
3. Checksum — MODBUS RTU uses algorithm CRC16.

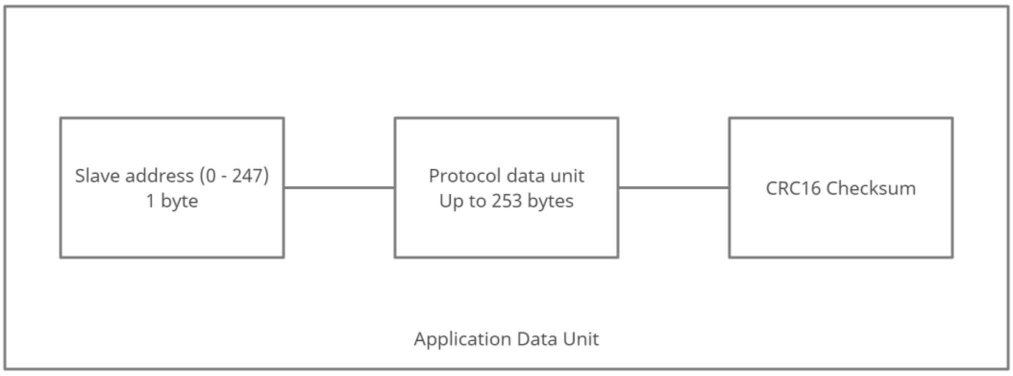


Figure 2. MODBUS protocol ADU representation

2.4 MODBUS Registers and Functions

2.4.1 Types of Registers

A MODBUS slave device includes four data storages; each can hold up to 9999 variables, also known as registers. Using registers, the master device can send the data to the slave device or read slave device data. The primary purpose of registers is to control and collect data from the specific device on the MODBUS network.

1. Discrete Output Coils (DO) – 1-bit registers can be written or read by the master device. The register number is defined as follows: start with 0 followed by four digits representing the entity's location. If the master device uses extended register notation, five digits are specified. Modbus guidelines define 1 for a given Coil as an ON state, and the same guidelines define the value of 0 for a given Output Coil as an OFF state.
2. Discrete Input Contacts (DI) – 1-bit read-only registers. The register number is defined as follows: start with digit one followed by four digits representing the entity's location. If the master device uses extended register notation, five digits are specified. Modbus guidelines define 1 for a given Coil as an ON state, and the same guidelines define the value of 0 for a given Output Coil as an OFF state.
3. Analog Input Register (AI) – 16-bit read-only registers. The register number is defined as follows: start with digit three followed by four digits representing the entity's location. If the master device uses extended register notation, five digits are specified.
4. Analog Output Register (AO) – 16-bit registers can be written or read by the master device. The register number is defined as follows: start with digit four followed by four digits representing the entity's location. If the master device uses extended register notation, five digits are specified.

2.4.2 Modbus Function Examples

The list of functions below is not a complete list of functions defined by the Modbus Application Protocol[3], it only includes examples and functions used to implement the goal defined by the thesis description.

[3] Modbus application protocol specification V1.1b3.
[*] N is the number of Outputs divided by 8; if the remainder is not 0, N = N+1

- Read Coils – Function code 01 is used to read from 1 to 2000 contagious coils in a remote device. PDU consists of function code (1 byte), Starting address (2 bytes), Quantity of Coils (2 bytes).
    - Response PDU consists of Function code 0x01 (1 byte), Byte count N* (1 Byte), Coil Status Values (n Bytes, where n = N or N+1)

- Read Discrete Inputs – Function code 02 is used to read from 1 to 2000 contagious status of discrete inputs in a remote device. PDU consists of function code (1 byte), Starting address (2 bytes), Quantity of Discrete inputs (2 bytes).
    - Response PDU consists of Function code 0x02 (1 byte), Byte count N* (1 Byte), Input Status Values (n Bytes, where n = N or N+1)
- Read Holding Registers – Function code 03 is used to read from 1 to 125 contagious blocks of holding registers in a remote device. PDU consists of function code (1 byte), Starting Address (2 bytes), Quantity of Registers (2 bytes).
    - Response PDU consists of Function code 0x03 (1 byte), Byte count N* (1 Byte), Register Values (N* bytes)
- Read Input Registers - Function code 04 is used to read from 1 to 125 contagious input registers in a remote device. PDU consists of function code (1 byte), Starting Address (2 bytes), Quantity of Registers (2 bytes).
    - Response PDU consists of Function code 0x04 (1 byte), Byte count N* (1 Byte), Input Register Values (N* bytes)
- Write Single Coil – Function code 05 is used to write the state of a single output to either ON or OFF state in a remote device. PDU consists of function code (1 byte), Slave Device Address (2 bytes), to request Coil to switch to on state master device specifies value 0x0000 or 0xFF00 to request slave device to switch Coil to OFF state (2 bytes).
    - Response PDU consists of Function code 0x04 (1 byte), Slave Device Address (2 bytes)

- Write Single Register – Function code 06 writes a single holding register in a remote device. PDU consists of function code (1 byte), Slave Device Address (2 bytes), value in a range of 0x0000 to 0xFFFF (2 bytes)
    - Response PDU consists of Function code 0x06 (1 byte), Register Address (2 bytes), Register Value (2 bytes)

3 IOT-TICKET PLATFORM

3.1 About IoT-Ticket platform

Wapice Oy describes the IoT-Ticket platform as a complete tool suite that allows customers to build web, mobile, cloud, and reporting Internet of Things-related applications.

The IoT-Ticket platform includes big data analytics support and user-friendly tools.

The IoT-Ticket improves the Internet of Things experience by allowing users to create IoT applications in minutes without additional software development or expenses.[4]

3.2 IoT-Ticket Platform Features

IoT-Ticket is a user-friendly web-based IoT platform that includes a customizable dashboard for personal computers and mobile phones, data analysis, device event automation.

While Wapice Oy targets IoT-Ticket for business solutions, studies can base on the tools provided by the platform. Software implementation of Modbus master for Linux can be modified to use any other IoT platform.

[4]Spice IoT-TICKET product description webpage.

4 ARDUINO MODBUS SLAVE DEVICES

4.1 Arduino Uno and Arduino Mega 2560

Arduino Uno[5] is a consumer-grade open-source development microcontroller board based on ATmega328P microchip[6]. This project includes two Arduino Uno development boards that are functioning as Modbus RTU Slave devices as humidity and temperature measuring devices.

Low price, fast development cycle, and popularity might make this development board a good choice for home automation.

Arduino Mega 2560 is a consumer-grade open-source development microcontroller board based on the ATmega2560 microchip[7]. One Arduino Mega 2560 development board functions as a Modbus RTU Slave device as an RGB LED controller.

This board has a higher price than Arduino Uno. However, it has more processing power and additional serial interfaces, digital and analog IO allowing it to be more versatile than Arduino Uno.

4.2 MAX485 Based Serial to RS-485 Module

Generic serial to the RS-485 module allows development boards to connect to the Modbus network, which by default is not supported by either of Arduino development boards. The modules used in this project utilize the Maxim Integrated MAX485ESA[8] chip.

The module can is usable with any development board which has a serial interface. However, it is optimal to design and manufacture custom serial to RS-485. The module mentioned here includes terminating resistor, which might cause issues in

[5] Arduino Uno datasheet.
[6] ATmega328P datasheet.
[7] ATmega2560 datasheet.
[8] MAX485ESA datasheet.

Modbus networks with many devices, as only one termination resistor should be in the network.

The MAX485ESA chip by Maxim Integrated is a low-power, slew-rate-limited transceiver designed for half-duplex applications.

Depending on the RE and DE connection, the module works as a transmitter or receiver. The application of current to RE and DE pins switches the RS-485 module into transmit mode while lacking current switches to the retrieve mode.

By default, this module includes 120 Ω resistor R7 for serial line termination, as serial RS-485 line in this project includes multiple slave devices, removal of 120 Ω termination resistor R7 was necessary for two of the slave devices[9]. The removal of termination resistance in similar generic modules is a good solution for development purposes but not practical for developing a device dedicated to serving as a Modbus RTU slave device.[10]

[9] Wiring of RS485 Communications Networks.

[10] RS-485 basics: When termination is necessary, and how to do it properly.

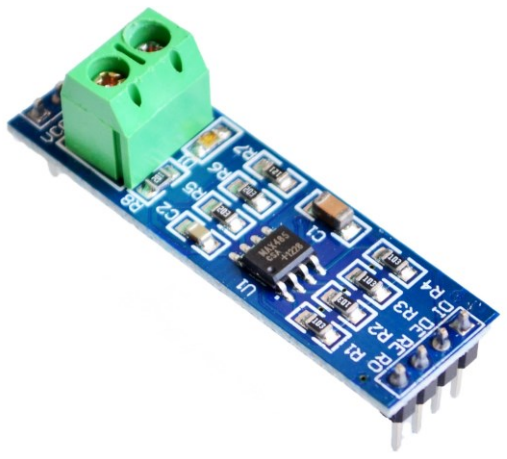


Figure 3. Generic Serial to RS-485 module

Pin connection for the implementation of connections within this project was the same for all devices as follows:

Vcc Pin-Connected to 5 Volt output of Arduino Board.

Ground pin – Connected to the ground pin of Arduino Board.

A pin – Connected to the A pin of USB to RS-485 converter.

B pin – Connected to the B pin of USB to RS-485 converter.

DE and RE pins – Connected to pin 3 of Arduino Board.

R0 pin – Connected to the RX0 pin of Arduino Board.

D1 pin – Connected to the TX0 pin of Arduino Board.

Table 1. MAX485 Technical Specification

# Tx/Rx	1Tx + 1Rx
Duplex	Half

$V_{(supply)}$ (V)	5
Data Rate (kbps) (min)	2500
I_{CC} (mA) (Typ)	0.5
Oper. Temp. (°C)	-40 to +85

4.3 DHT22 Temperature and Humidity Sensor

DHT22 Temperature and Humidity Sensor is a popular consumer-grade sensor; it is common in most projects requiring temperature and Humidity measuring. The sensor outputs a digital signal and takes on average two seconds to take measurements. Small size, low power consumption, and long transmission distance allow this sensor to transmit data far from the development board, enabling placement in a different room.

Within the scope of this thesis, the DHT22 temperature and humidity sensor measures the ambient temperature and humidity; the master device collects measurements and uploads them to IoT-Ticket cloud as an example of variables that can be read from the Modbus Slave device and displayed in the IoT-Ticket dashboard.

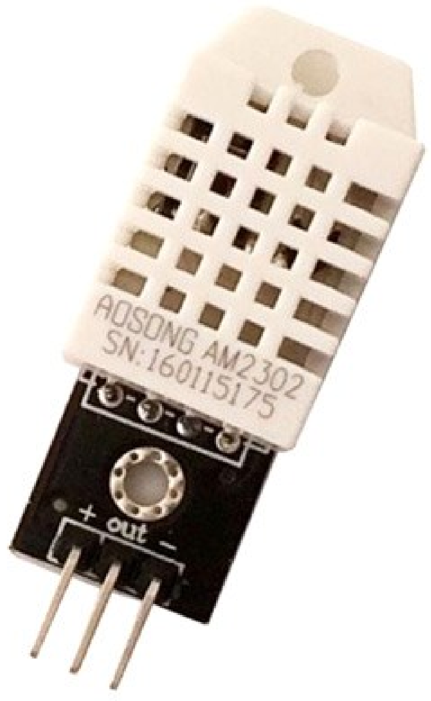


Figure 4. DHT22(AM2302) Sensor

Table 2. DHT22 Technical Specification[11]

Model	DHT22
Power Supply	3.3-6V DC
Output signal	Digital signal via single-bus
Operating range	Humidity 0-100% RH Temperature -40~80 °C
Accuracy	Humidity +-2% RH (Max +-5% RH) Temperature <+-0.5 °C
Resolution of sensitivity	Humidity 0.1% RH Temperature 0.1 °C
Repeatability	Humidity +-1% RH Temperature +-0.2 °C
Sensing period	Average: 2s
Interchangeability	Fully interchangeable

[11] Digital-output relative humidity & temperature sensor/module DHT22.

4.4 KY-009 RGB LED Module

KY-009 is an RGB full-color LED module for Arduino[12], capable of emitting a range of colors by mixing red, green, and blue light. The amount of each color is adjusted using PWM.

This RGB LED module can show the possibility of writing the values given in the IoT-Ticket Dashboard to the slave devices.

4.5 Modbus RTU Arduino Library

Modbus RTU Arduino Library is a part of a project authored by Samuel Marco I Armengol under the GNU License and publicly available on GitHub.[13]

This library implements a serial Modbus protocol for Arduino devices. Within the scope of this thesis, Arduino development boards use this library to interact as Slave devices in the Modbus Network.

4.6 SimpleDHT Library

The SimpleDHT Arduino library is a part of a project authored by Winlin under the MIT License and publicly available on GitHub.[14]

This library implements interfacing with DHT22 and DHT11, which allows usage of DHT22 sensor on Arduino Uno Modbus Slave devices.

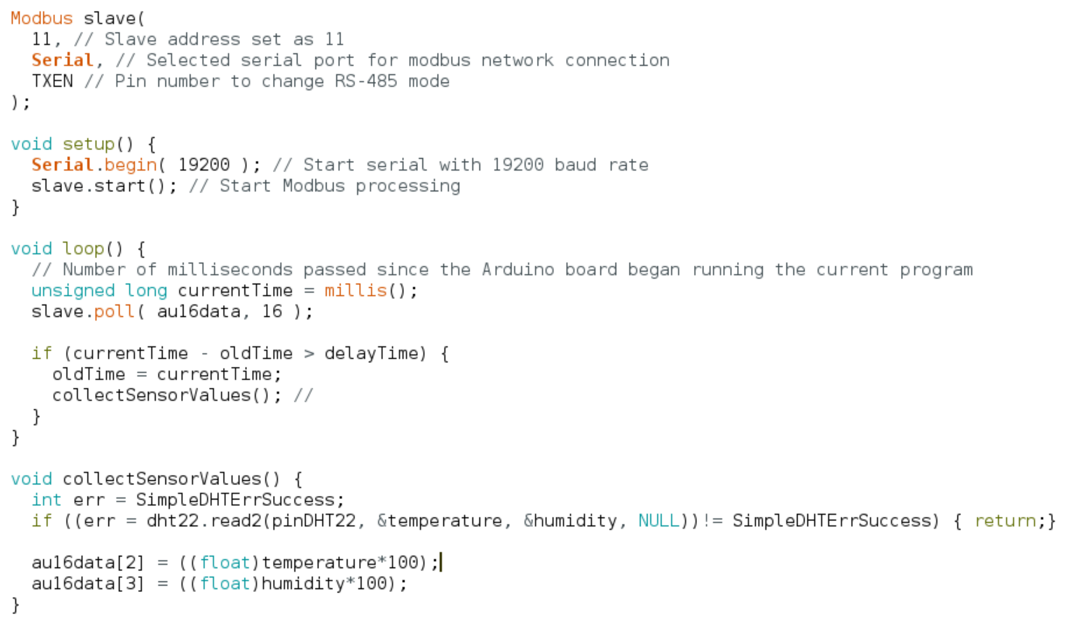
4.7 Source Code for Arduino Modbus Slave Devices

Arduino Uno Modbus slave devices share the source code for the program; the only difference between devices is the slave address. The first Arduino Uno Slave Device uses the number ten as its address, and the second Slave Device uses 11 as its address.

[12] KY-009 Datasheet.
[13] Modbus-Master-Slave-for-Arduino.
[14] SimpleDHT.

Arduino development boards utilize the ModbusRtu library to allow boards to serve as a Modbus Slave Device. Each of the Arduino boards uses pin three to change the RS-485 module mode; this pin connects to pins DE and RE of serial to RS-485 Module.

The Arduino Uno development board code includes the SimpleDHT library to collect measured data from the DHT22 sensor.



```
Modbus slave(
  11, // Slave address set as 11
  Serial, // Selected serial port for modbus network connection
  TXEN // Pin number to change RS-485 mode
);

void setup() {
  Serial.begin( 19200 ); // Start serial with 19200 baud rate
  slave.start(); // Start Modbus processing
}

void loop() {
  // Number of milliseconds passed since the Arduino board began running the current program
  unsigned long currentTime = millis();
  slave.poll( au16data, 16 );

  if (currentTime - oldTime > delayTime) {
    oldTime = currentTime;
    collectSensorValues(); //
  }
}

void collectSensorValues() {
  int err = SimpleDHTErrSuccess;
  if ((err = dht22.read2(pinDHT22, &temperature, &humidity, NULL))!= SimpleDHTErrSuccess) { return;}

  au16data[2] = ((float)temperature*100);
  au16data[3] = ((float)humidity*100);
}
```

Figure 5. Arduino Uno Modbus Device Slave code snippet

DHT22 average measurement time is two seconds. The 2-second delay helps to avoid complications induced by the mechanics of the DHT22 sensor; Arduino Uno Modbus Slaves update humidity and temperature values in the register during the measurement collection from the DHT22 sensor.

One of the limitations of the Modbus network is the inability to pass floating-point values directly. Temperature and humidity values were multiplied by 100, creating an integer value, divided by 100 by the Modbus Master device, allowing transfer of floating-point values.

Arduino Mega 2560 Modbus Slave Device has a similar codebase. However, instead of writing data to the registers, the development board reads the register values and writes them as PWM values to the pins 5, 6, and 7, corresponding to Red, Green, and Blue components of the RGB LED module. Arduino Mega 2560 development board has number 12 as slave address.

```
void setup() {
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
  pinMode(ledpin, OUTPUT);
  Serial.begin( 19200 ); // baud-rate at 19200
  slave.start();
}

void loop() {
  slave.poll( au16data, 16 );
  if(redVal != au16data[1]) {
    redVal = au16data[1];
    setColor(redVal, greenVal, blueVal);
  }

  if(greenVal != au16data[2]) {
    greenVal = au16data[2];
    setColor(redVal, greenVal, blueVal);
  }

  if(blueVal != au16data[3]) {
    blueVal = au16data[3];
    setColor(redVal, greenVal, blueVal);
  }
}

void setColor(int redValue, int greenValue, int blueValue) {
  analogWrite(redPin, redValue);
  analogWrite(greenPin, greenValue);
  analogWrite(bluePin, blueValue);
}
```

Figure 6. Arduino Mega 2560 Modbus Device Slave code snippet

5 PYTHON-BASED APPLICATION

5.1 Application Description

The Python[15] application is supposed to serve as a Modbus Network Master device by connecting to the network using the USB to RS-485 module. Currently, the codebase allows the connection of multiple Modbus Network Segments. However, within the scope of this thesis, only one segment is present.

The application consists of four parts and two external libraries, MinimalModbus authored by Jonas Berg, licensed under Apache License[16] and IoT-Ticket library developed by Wapice Ltd, licensed under MIT License[17].

The application loads the device list from JSON files in 'devices' folder in the same directory as the core file of the Python project, creating and filling the class with the information provided by the JSON file. The JSON files include device name, the serial port of Modbus Network in which device is located, register information, datatypes, units.

The software considers each register as separate IO; the application creates a task from the IO information. The application uses tasks to interact with the registers and handle reading and writing from and to registers.

The application iterates with adjustable time intervals, each iteration reduces the time until task execution, and when the time until execution equals zero, the application starts processing the task. Upon the start of task execution, the application selects action depending on the register type. Current IoT-Ticket cloud data is being downloaded with adjustable intervals, defaulted by 5 seconds.

Tasks related to Discrete Inputs and Input Registers are executed independently from stored Cloud Data. Discrete Inputs and Input Software reads all registers once

[15] Python 3.9.5 documentation.
[16] MinimalModbus.
[17] IoTTicket-PythonLibrary.

the time until the execution is zero. The task processor compares the current value in Discrete Input or Coil. If values have changed, new values are packed into data node packages and uploaded to according devices to the IoT-Ticket cloud storage at the end of application iteration. It is possible to configure the task to upload data without comparing.

Tasks related to Coils and Holding Registers application are considered as Cloud Data Sensitive. The execution of such tasks only occurs when the time until execution is zero and Cloud Data was just loaded. Task processing consists of reading the current value in Cloud Data, the current value in Coil or Holding Register, and writing value from the Cloud Data to the device.

The application tracks the current state of Coils and Holding registers by rereading the values from slave devices and uploading them to the cloud with the appendix either "_coil_state” or “_holding_register_state."

5.2 USB to RS-485 Module

The personal computer, which serves as Modbus Device Master, is connected to the Modbus Network using a generic USB to RS-485 module. USB to RS-485 module supports half duplex transmission, 1200 meters transmission rate at transmission speed of 9600b/s[18].

Terminal A of the module connects to the RS-485 modules of Arduino Slave Devices terminal A, and terminal B connects to the RS-485 module terminal B of Arduino Slave Devices.

[18] Usb to serial chip CH340 data sheet.

Figure 7. Generic USB to RS-485 adapter

5.3 MinimalModbus library

The MinimalModbus library, developed by Jonas Berg, is a Python driver for Modbus RTU and ASCII protocols, supporting serial connections using RS-485, RS-232, or USB to serial adapters. The library is licensed under Apache License, version 2.0.

The MinimalModbus library includes Instrument class to communicate to instruments, also known as slaves within Modbus RTU network.

The library includes implementation of Modbus functions: Read Coils, Read Discrete Inputs, Read Holding Registers, Read Input Registers, Write Single Coil, Write Single Register, Write Multiple Coils, and Write Multiple Registers.

It is possible to transmit or receive float values and ASCII strings using this library. However, the target device might not support it.

5.4 IoT-Ticket Library

The IoT-Ticket library for Python is a library developed by Wapice Ltd and released under the MIT License. The library uses IoT-Ticket REST API.

It provides a set of functions for interaction with the IoT-Ticket platform, allowing data transmission between the platform and application without writing a custom library.

The methods of the library include functions for information fetching, data reading, and writing. The application implemented within the scope of this thesis utilizes read and write data functions of the IoT-Ticket library.

5.5 Application class structure

The application operates using array of large class named Device class, that includes name of the slave device, description of the slave device, Instrument class (Implemented in the MinimalModbus library and used for communication with the slave devices),

serial port dedicated to the Modbus network segment, slave device address, array of sub-class Raw IO info that stores the information about slave device registers, and array of sub-class Task Data that stores the information required for task processor.

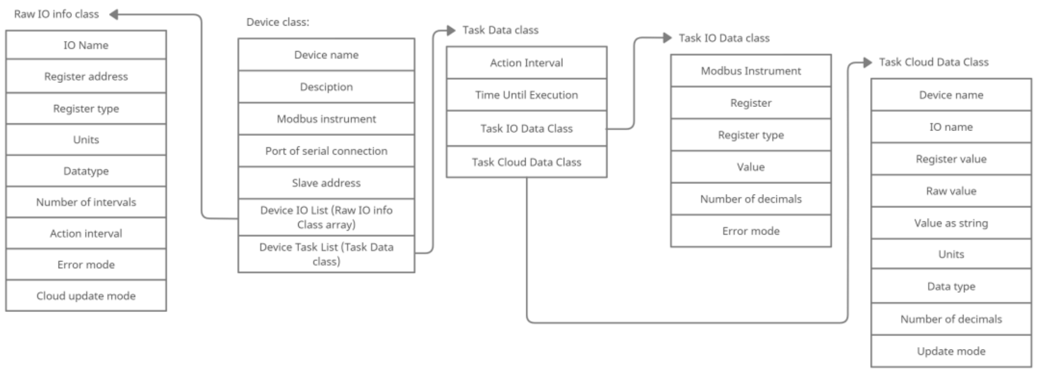
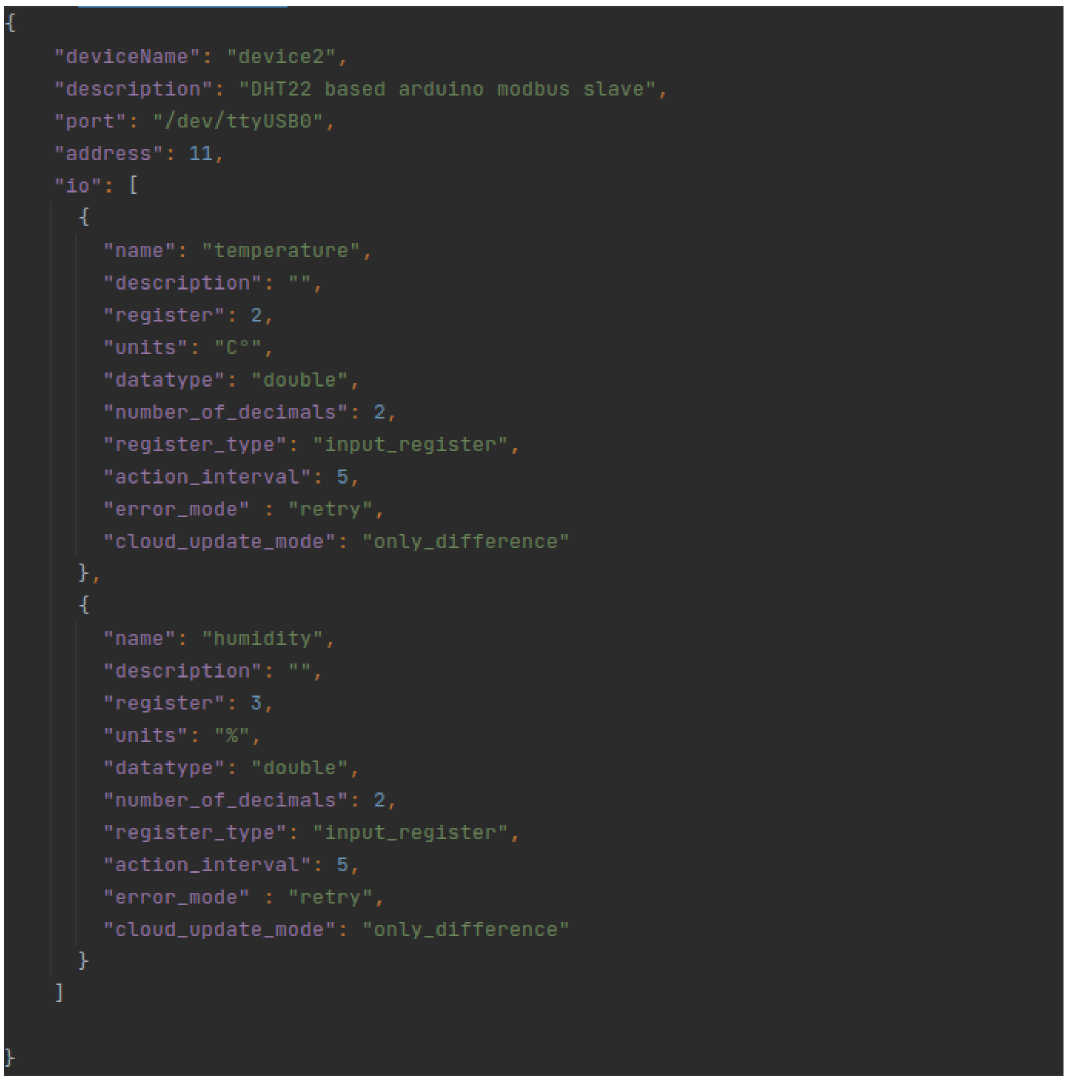


Figure 8. A complete representation of application class structure

5.6 Device Information Storage and Loading Procedure

As the Modbus master device is unaware of the number of slave devices in the Modbus network and their specifications, the user must create a JSON file with the information about the device. The JSON file must include the name of the slave device, the serial port with required Modbus network segment, slave device address, and list of IO on the specified device. IO is a name that would describe a package that includes register information, such as name, register address, data type, number of decimals, register type, and some of the other information that task processor and IoT-Ticket cloud require.



```
{
    "deviceName": "device2",
    "description": "DHT22 based arduino modbus slave",
    "port": "/dev/ttyUSB0",
    "address": 11,
    "io": [
      {
        "name": "temperature",
        "description": "",
        "register": 2,
        "units": "C°",
        "datatype": "double",
        "number_of_decimals": 2,
        "register_type": "input_register",
        "action_interval": 5,
        "error_mode" : "retry",
        "cloud_update_mode": "only_difference"
      },
      {
        "name": "humidity",
        "description": "",
        "register": 3,
        "units": "%",
        "datatype": "double",
        "number_of_decimals": 2,
        "register_type": "input_register",
        "action_interval": 5,
        "error_mode" : "retry",
        "cloud_update_mode": "only_difference"
      }
    ]

}
```

Figure 9. Example of JSON file with Modbus slave device information

The JSON file includes data required for Task processor, Modbus network interaction, and information required for interaction with IoT-Ticket platform. The JSON file consists of:

- deviceName – The name of the Modbus slave device, under this name, the data of the slave device will appear on the IoT-Ticket platform.
- description – Used for the description of the device, however unused in the application or IoT-Ticket platform.
- port – Location of the interface within the master device that holds the Modbus network segment containing the slave device.
- address – Address of the slave Modbus device within the Modbus network.

- io – List of the information for each register, defined as:
    - name – Name of the register, used for data storage on the IoT-Ticket platform.
    - description – Used for the description of the register, however unused in the application or IoT-Ticket platform.
    - register – address of the register within the Modbus slave device.
    - units – Used only to specify units of the register within the IoT-Ticket platform.
    - datatype – Used for specification of the datatype to store on the IoT-Ticket platform.
    - number_of_decimals – Required for specification of a number of decimals of the float values. 0 if the register contains integer or boolean value.
    - register_type – Type of the register. The task processor requires this to perform a correct operation. It can be either of four register types.
    - action_interval – Time in seconds between the execution of the task.
    - error_mode – Mode of error handling for the given register. "Retry" – will instruct the task processor to retry to send or receive data from the register upon encountering exceptions. "Skip" – will instruct the task processor to ignore the exception and switch to the next task.
    - cloud_update_mode – As resources of the IoT-Ticket platform are limited, it is possible to instruct the task processor to either upload values only when it is different from currently stored in the IoT-Ticket or update the values upon every execution of the task.

Upon the startup of the application, the device information loader loads the device information from the JSON file into the device class, saving all the information from the file in the application memory.

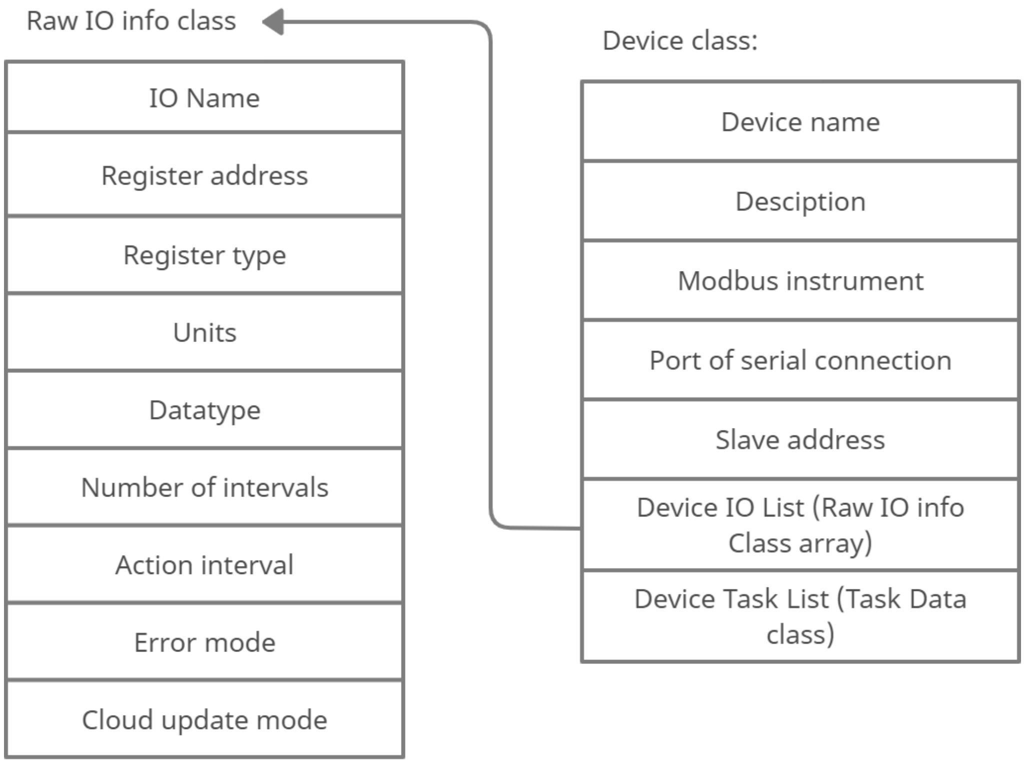
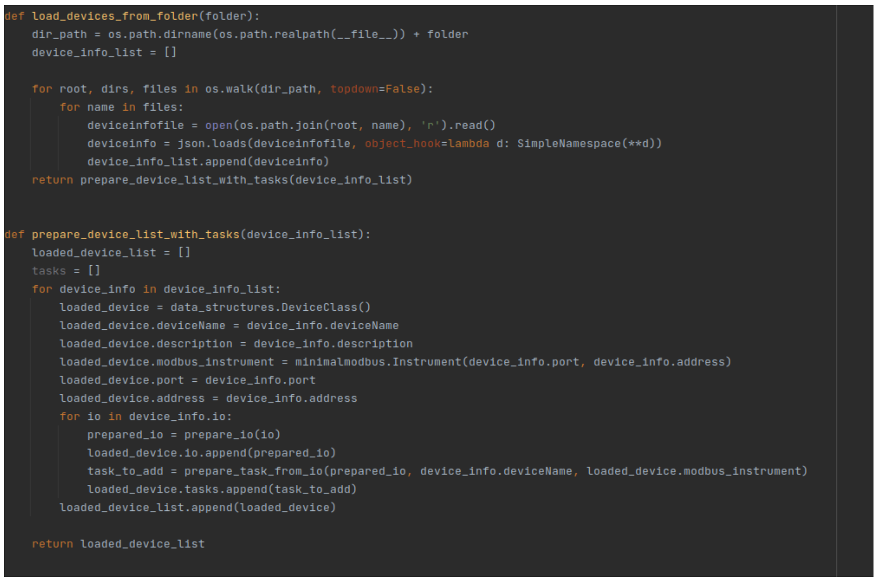


Figure 10. Device class and Raw IO info class representation.

As the raw information from the JSON files is loaded, the device loader starts processing the information and builds a task list based on a list of IO specified by the file. Once loading is complete, tasks are being created for every IO located in the IO list of the given device.


```python
def load_devices_from_folder(folder):
    dir_path = os.path.dirname(os.path.realpath(__file__)) + folder
    device_info_list = []

    for root, dirs, files in os.walk(dir_path, topdown=False):
        for name in files:
            deviceinfofile = open(os.path.join(root, name), 'r').read()
            deviceinfo = json.loads(deviceinfofile, object_hook=lambda d: SimpleNamespace(**d))
            device_info_list.append(deviceinfo)
    return prepare_device_list_with_tasks(device_info_list)


def prepare_device_list_with_tasks(device_info_list):
    loaded_device_list = []
    tasks = []
    for device_info in device_info_list:
        loaded_device = data_structures.DeviceClass()
        loaded_device.deviceName = device_info.deviceName
        loaded_device.description = device_info.description
        loaded_device.modbus_instrument = minimalmodbus.Instrument(device_info.port, device_info.address)
        loaded_device.port = device_info.port
        loaded_device.address = device_info.address
        for io in device_info.io:
            prepared_io = prepare_io(io)
            loaded_device.io.append(prepared_io)
            task_to_add = prepare_task_from_io(prepared_io, device_info.deviceName, loaded_device.modbus_instrument)
            loaded_device.tasks.append(task_to_add)
        loaded_device_list.append(loaded_device)

    return loaded_device_list
```

Figure 11. Code snippet of device loader

The task contains the time until execution, task execution interval, and information about the Modbus slave device and IoT-Ticket platform. Essentially, the task class contains information required for the transaction of information between the Modbus slave device and the IoT-Ticket platform.

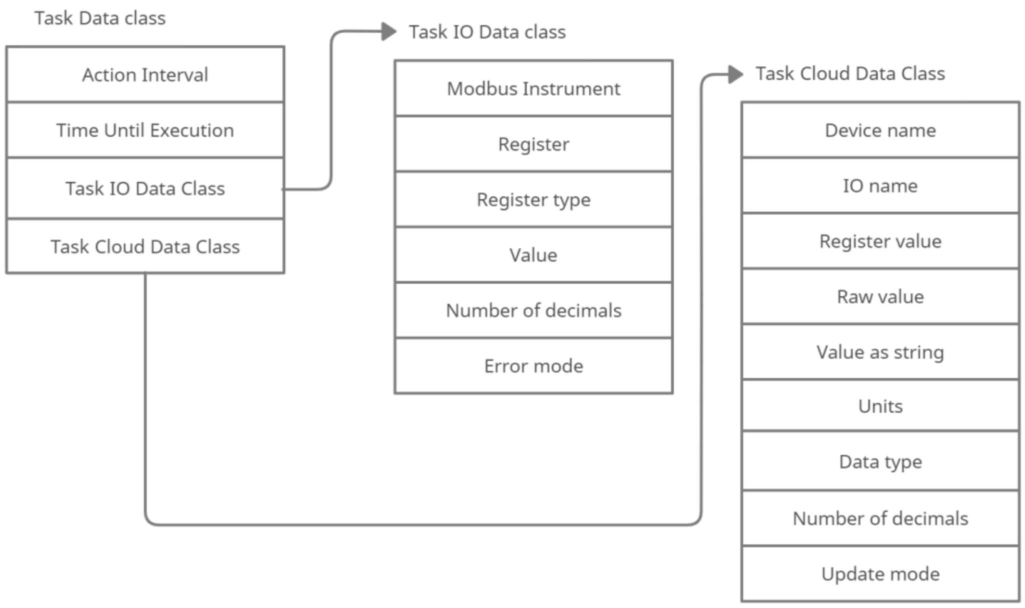


Figure 12. Task Data class structure

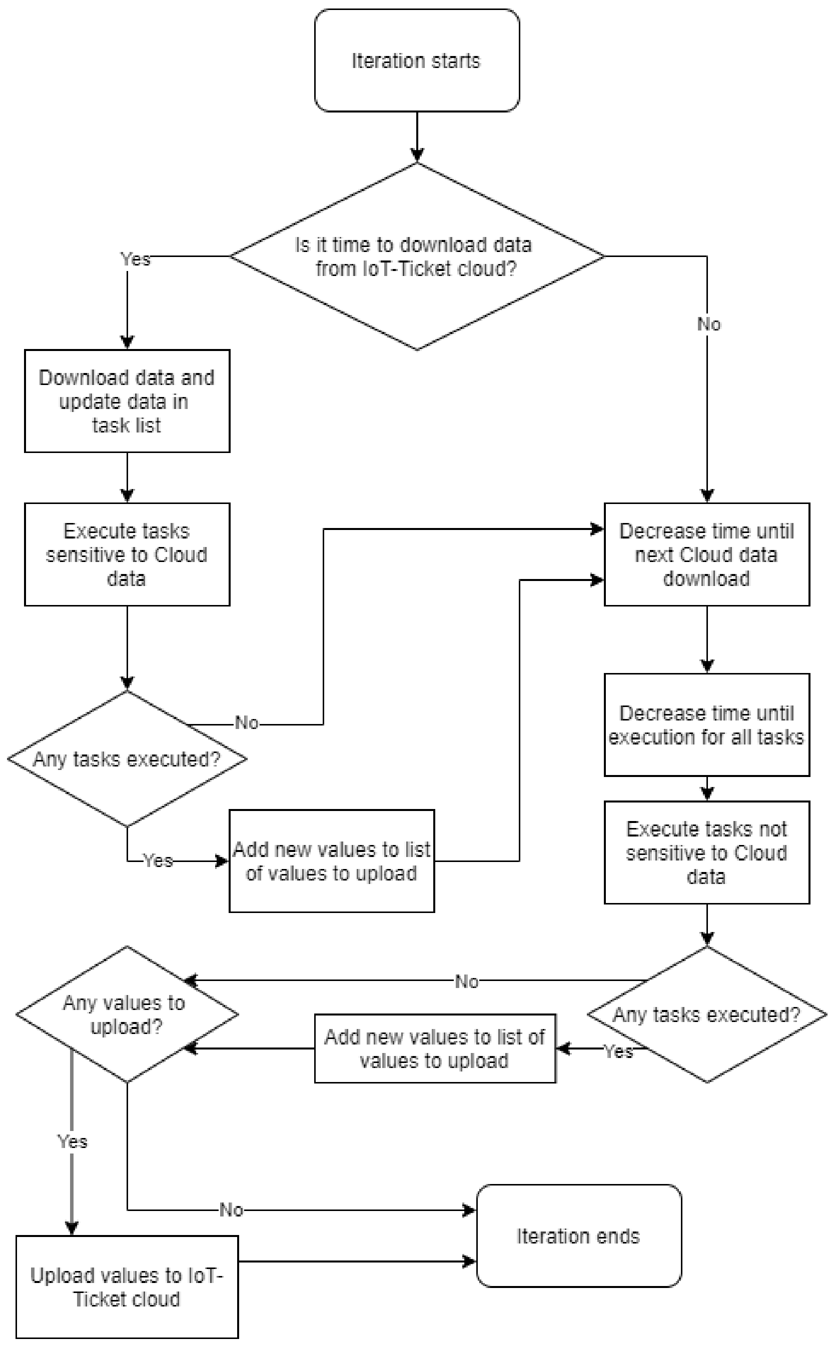
5.7 Iteration over Tasks

Once the application loads device information and prepares a list of tasks, it downloads the most recent data from the IoT-Ticket and assigns information from the cloud to each task.

Each iteration, a function is called to decrease the time until a new batch of data downloads from the cloud. The execution of task starts when the time until execution equals zero. After the execution, the time until the subsequent execution equals pre-defined in JSON file time.

Tasks based on interaction with Coils and Holding registers are considered as cloud data sensitive. The primary function of Coils and Holding registers is to input information into the Modbus slave devices; actual data from the IoT-cloud storage must be present.

Tasks based on interaction with Discrete inputs and Input registers do not require actual IoT-cloud data, as the Modbus master device only uploads it to the IoT-cloud storage.



Iteration starts
Is it time to download data from IoT-Ticket cloud?
Yes
No
Download data and update data in task list
Execute tasks sensitive to Cloud data
Any tasks executed?
No
Yes
Add new values to list of values to upload
Decrease time until next Cloud data download
Decrease time until execution for all tasks
Execute tasks not sensitive to Cloud data
Any tasks executed?
No
Yes
Add new values to list of values to upload
Any values to upload?
Yes
No
Upload values to IoT-Ticket cloud
Iteration ends

Figure 13. Modbus master application iteration flowchart

5.8 Task handling

The task execution starts from checking the time until execution; if that value is not zero, the task processor is skipping the task. Otherwise, the task processor selects action depending on the type of register specified in the task.

The task execution for the Discrete Input and Input Register involves reading data from the slave device. The task processor compares the data fetched from the slave device to the stored data from the same register. If the fresh data and the stored data are different or the cloud update mode "Always" is present, the task processor will prepare the data node package to upload to the IoT-Ticket cloud.

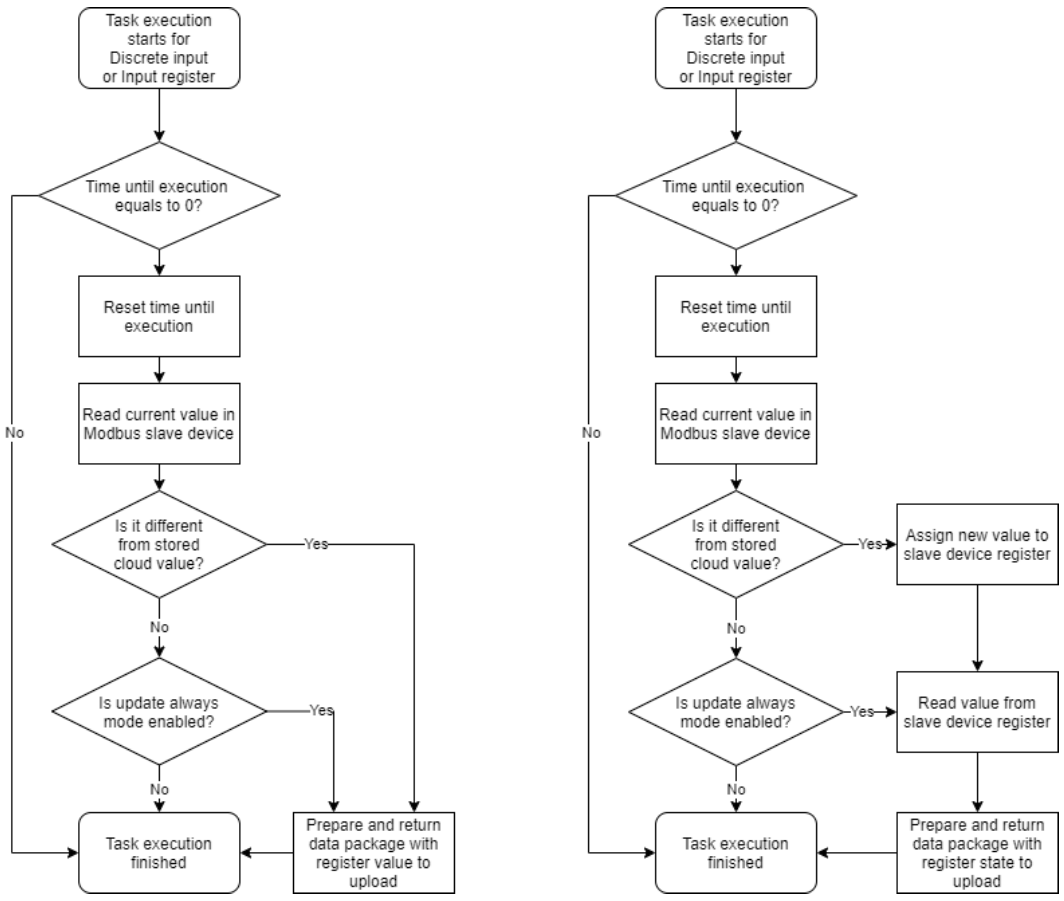

Task execution starts for Discrete input or Input register
Time until execution equals to 0?
Reset time until execution
Read current value in Modbus slave device
Is it different from stored cloud value?
Is update always mode enabled?
Yes
No
Task execution finished
Prepare and return data package with register value to upload
Task execution starts for Discrete input or Input register
Time until execution equals to 0?
Reset time until execution
Read current value in Modbus slave device
Is it different from stored cloud value?
Assign new value to slave device register
Is update always mode enabled?
Read value from slave device register
Task execution finished
Prepare and return data package with register state to upload

Figure 14. Task execution flowchart

The task execution for the Coil and Holding Registers consists of reading data from the slave device. The task processor compares the data fetched from the slave device to the data downloaded from the IoT-Ticket cloud. If there is a difference, the task processor assigns register in the slave device the value from IoT-Ticket cloud. To ensure the data was assigned correctly, the task processor reads the value from the slave device and prepares the data node package with the suffix "_state." The cloud update mode "Always" will upload the current value of the slave device register in a data node package with the suffix "_state."

Once all actions are executed within the task, time until execution for a given task is set the same value as the time specified within configuration. If the task has an unknown register type, the task is assigned the lowest priority; the task processor will not process it in the future.

5.9 Modbus Network Handling

The Modbus master Python application uses the MinimalModbus library to interact with the Modbus RTU network, USB to RS-485 module establishes a physical connection to the network. The custom library prevents critical exceptions by retrying reading or writing attempts upon encountered exceptions. The most commonly encountered exception is connection timeout. As RS-485 has a low tolerance towards delays and a poor-quality cable was used during development, shielded cables can prevent this exception from happening.

5.10 Interaction with IOT-Ticket Platform

The Modbus master Python application uses the IoT-Ticket library to receive actual data from the cloud storage, encapsulates data node packages, and sends them back to the IoT-Ticket cloud storage.

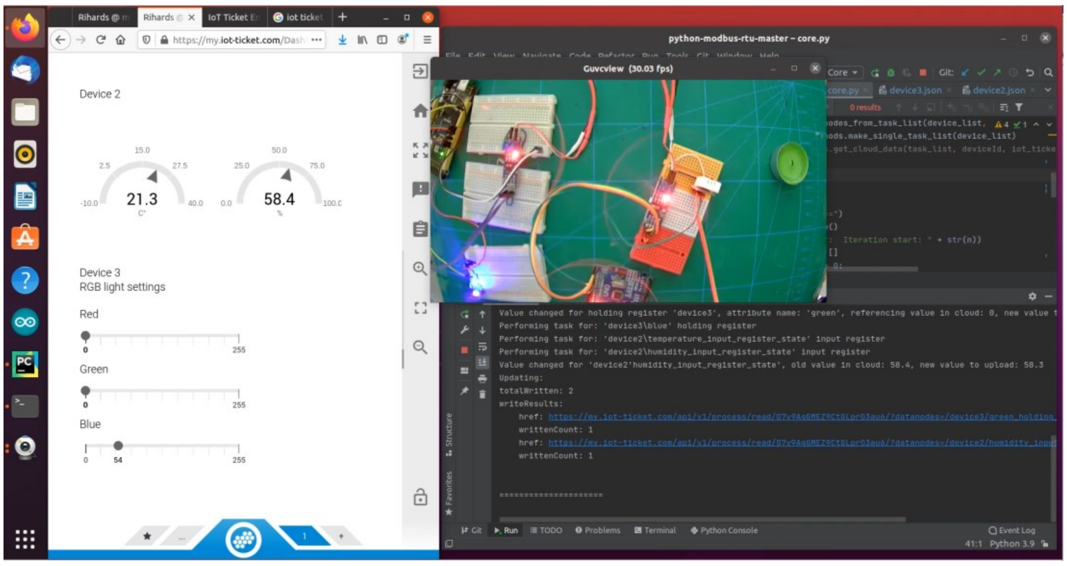
The application creates IoT-Ticket data node packages that include the name of the variable, the value of the variable, the data type, and optionally, units at the end of task execution. The application uploads the values at the end of the iteration. The application tries to minimize interaction with the IoT-Ticket platform as it does not have infinite resources. Every user has a quota of data that can be sent and stored at the IoT-Ticket platform.

6 TESTING

After the development and implementation of the network, the system testing followed. The system can send and receive data from the IoT-Ticket platform in the current state, including variables with boolean, floating-point, and integer data types, and assign them to the Coils and Holding register in the slave devices. The system can read values from the Input registers and Discrete inputs and upload them to the IoT-Ticket platform.

Tests were performed using three Arduino boards; two included the DHT22 sensor, and one included the RGB Led module.

During the tests, the capability of reading the values to the RGB Led module was tested. Humidity and temperature tests from the DHT22 were also successful.



```
Value changed for holding register 'device3', attribute name: 'green', referencing value in cloud: 0, new value t
Performing task for: 'device3\blue' holding register
Performing task for: 'device2\temperature_input_register_state' input register
Performing task for: 'device2\humidity_input_register_state' input register
Value changed for 'device2'humidity_input_register_state', old value in cloud: 58.4, new value to upload: 58.3
Updating:
totalWritten: 2
writeResults:
    href: https://my.iot-ticket.com/api/v1/process/read/07v9AqGMEZ9CtGLpr03au6/?datanodes=/device3/green_holding_
    writtenCount: 1
    href: https://my.iot-ticket.com/api/v1/process/read/07v9AqGMEZ9CtGLpr03au6/?datanodes=/device2/humidity_input
    writtenCount: 1


====================
```

Figure 15. Screenshot from the system demonstration video

7 CONCLUSIONS

The implementation and development of the system, including Modbus RTU protocol and IoT-Ticket platform, can be considered a success. The system can communicate with the Modbus network and utilize IoT-Ticket storage.

The system can be used for domestic or study purposes; it might be unreliable for implementation in industrial production. It can be part of the study process as it includes the standard machine-to-machine protocol and supports custom devices.

It is possible to improve the system by implementing different ways of passing the floating-point values, such as two integers. It is possible to implement string transmission. However, the target slave device must support it.

To make this system more viable for the industry or study purposes, it would be beneficial to develop more advanced exception handling, web-based control interface, support of different IoT platforms, and support of network configuration during the system execution.

In the current state, home environments can utilize the system as an alternative to wireless IoT networks.

REFERENCES

Full guide to serial communication protocol and our RS-485. 2019. Accessed 26.04.2021. https://www.maximintegrated.com/en/design/technical-documents/app-notes/3/3884.html

Modbus FAQ: About The Modbus Organization. 2005. Accessed 26.04.2021. https://www.modbus.org/faq.php

Modbus application protocol specification V1.1b3. 2012. Accessed 26.04.2021. https://modbus.org/specs.php

Spice IoT-TICKET product description webpage. 2019. Accessed 26.04.2021. https://www.wapice.com/products/iot-ticket

Arduino Uno datasheet. 2013. Accessed 26.04.2021. https://www.farnell.com/datasheets/1682209.pdf

ATmega328P datasheet. 2015. Accessed 26.04.2021. https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

ATmega2560 datasheet. 2014. Accessed 26.04.2021. https://ww1.microchip.com/downloads/en/devicedoc/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf

MAX485ESA datasheet. 2014. Accessed 26.04.2021. https://datasheets.maximintegrated.com/en/ds/MAX1487-MAX491.pdf

Wiring of RS485 Communications Networks. 2020. Accessed 26.04.2021. https://www.se.com/ww/en/faqs/FA221785/

RS-485 basics: When termination is necessary, and how to do it properly. 2021. Accessed 26.04.2021. https://e2e.ti.com/blogs_/b/analogwire/posts/rs-485-basics-when-termination-is-necessary-and-how-to-do-it-properly

Thomas Liu. 2011. Digital-output relative humidity & temperature sensor/module DHT22. Accessed 26.04.2021. https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf

KY-009 Datasheet. 2017. Accessed 26.04.2021. https://datasheetspdf.com/datasheet/KY-009.html

Modbus-Master-Slave-for-Arduino. 2014. Accessed 26.04.2021. https://github.com/smarmengol/Modbus-Master-Slave-for-Arduino

SimpleDHT. 2018. Accessed 26.04.2021. https://github.com/winlinvip/SimpleDHT

Python 3 documentation. 2018. Accessed 26.04.2021. https://docs.python.org/3/

MinimalModbus. 2019. Accessed 26.04.2021. https://minimalmodbus.readthedocs.io/en/stable/

IoTTicket-PythonLibrary. 2016. Accessed 26.04.2021. https://github.com/IoT-Ticket/IoTTicket-PythonLibrary

Usb to serial chip CH340 data sheet. 2017. Accessed 26.04.2021. https://cdn.sparkfun.com/datasheets/Dev/Arduino/Other/CH340DS1.PDF