

University of Cyprus
Computer Science Department

PRESENTATION PLAN
INDIVIDUAL DISSERTATION

May 2019

INDIVIDUAL DISSERTATION

Monitoring System for Photovoltaic Plants

Stefanos Ioannou

University of Cyprus



Computer Science Department

May 2019

University Of Cyprus
Computer Science Department

Monitoring System for Photovoltaic Plants



Supervisor

Marios D. Dikaiakos

Stefanos Ioannou

May 2019

Summary

Chapter 1	Purpose & System Architecture	5
	1.1 Project Purpose	
	1.2 ENEDI Project Correlation	
	1.3 General Structure	
	1.4 PV Technology & Characteristics	
Chapter 2	Microcontroller Collector	17
	2.1 Arduino Uno Rev3	
	2.2 Xbee Transmitter Module	
	2.3 Sensors	
	2.4 Software on the Microcontroller Collector	
	2.5 Power Supply	
Chapter 3	Single Board Computer Coordinator	24
	3.1 Raspberry Pi Model 3+	
	3.2 Xbee Receiver Module	
	3.3 Software on the SBC Coordinator	
Chapter 4	Cloud Virtual Machine	31
	4.1 Virtual Machine Characteristics	
	4.2 Grafana Framework	
Chapter 5	Prometheus Monitoring System	32
	5.1 Prometheus Concepts	
	5.2 SBC Coordinator Prometheus	
	5.3 Hierarchical Federation Prometheus	
	5.4 Prometheus Instrumenting	
Chapter 6	Architecture Justification	35
	6.1 Architecture Justification	
	6.2 Architecture Information	
Chapter 7	Optimization	37
	7.1 Alternative Infrastructure Architecture	
	7.2 Possible Add-Ons	
	References	41
	Appendix	43

Chapter 1

Purpose & System Architecture

1.1 Purpose of Project

Monitoring a photovoltaic system is considered a big challenge nowadays as the solar PV energy generation is growing. The increasing cost of thermal power generation has resulted in a shift to renewables. Many research centers as well as firms seek to possess a powerful monitoring structure to check the generated electricity from the solar modules in order to compute expenses and return-on-investments. Furthermore by maintaining an observation on performance metrics on various solar panel modules we can build an efficient PV module to increase the efficiency on solar power collection. Another use of this structure is to monitor home solar panels for personal usage. The implementation of the system is simple, by following the steps on the later sections, an engineer can easily build the infrastructure for home monitoring.

1.2 ENEDI Project Correlation

This project focuses on to build a fully functional cheap monitoring structure using microcontrollers and sensors that can gather photovoltaic metric. The proposed system is an alternative approach to the one that is used in ENEDI. The monitoring system designed in ENEDI project is quite expensive because the machines that are used have low observational error [N. Louloudis, page 33] and the Data Loggers are scientific expensive components. Particularly the Campbell Scientific CR3000 costs up to 3000\$ [Radwell International].

The modules that are used in the ENEDI project are DC Power & DC current & DC voltage Sensors, PV temperature sensor, Wind Speed Sensor, DataLogger, Ambient Temperature & Humidity Sensor, HTTP module, Global Pane of Array Irradiance [N.Louloudis ,page 34]. The proposed system must keep close to having the same functionality as that in the ENEDI with reduced expenditure. Moreover, the proposed alternative infrastructure has embedded microcontrollers and single board computers that can provide even more abilities to the researchers when monitoring the PV modules. Microcontrollers working together with single board computers make a very good combination of an error-resistant infrastructure and allows the organization that uses the system to maintain a high, consistent standard of data quality.

1.3 High Level Reference System Architecture

GUI layer

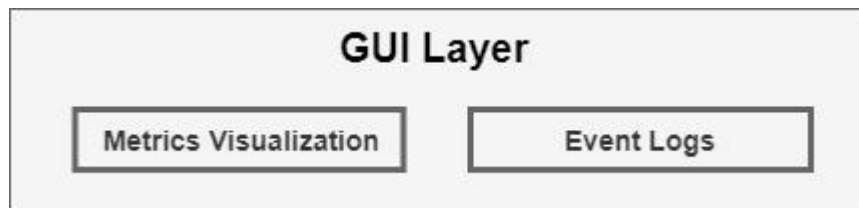


Figure 1.3.1: The GUI layer is divided into metrics visualization and event logs table.

Component	Description
Metrics Visualization	A graphical representation of the metrics that are gathered from the PV module using time plot charts. Visualization must be UI friendly and the user must be able to observe multiple charts in one window.
Event Logs	A text table with each record containing a timestamp with an event that happened regarding the status of the system. [ex. failure of a particular component at a particular time]. The event logs must be visualized in an elegant manner using the latest internet software technology.

The components of GUI layer use the following software and network protocols:

Component	Software / Programming Languages	Communication Protocols (Sorted following Internet protocol suite model)
Metrics Visualization	Grafana Dashboard	MAC / IPv4 / TCP / HTTP - 1.1 or 2.0
Event Logs	HTML/ CSS/ Javascript / Python 2.7	MAC / IPv4 / TCP / HTTP - 1.1 or 2.0



Figure 1.3.2: Visualizing data using Grafana Dashboard.

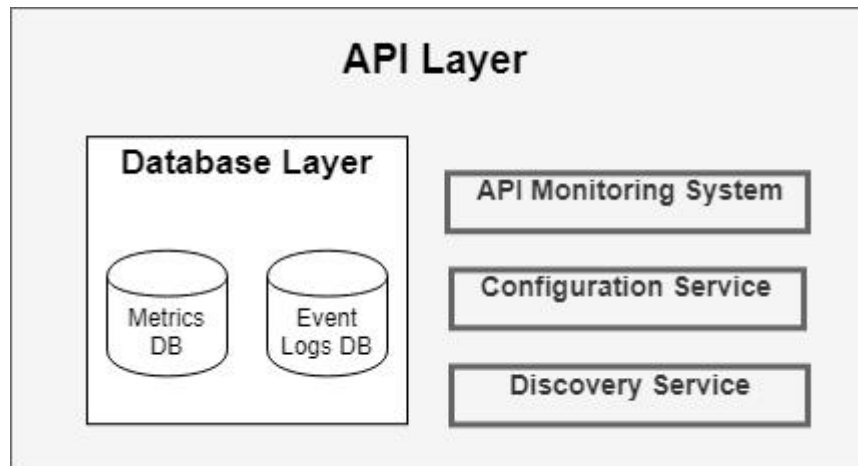


Figure 1.3.3: API layer is divided to the database layer, one system and two services.

The API layer demonstrates the organization of the components behind the system. The main component is the API monitoring system and secondary components are the configuration service and the delivery service.

All the subroutines of the API are seen in the Appendix Code section.

Component	Description
API Monitoring System	<p>Subroutine definitions:</p> <ul style="list-style-type: none"> ● Microcontroller Collector Script ● Database handler Script ● Python Pyramid Server Script ● SBC Coordinator Script <p>API is divided into the above modules for properly execution of the monitoring process. All the script modules are defined in the sections below.</p>
Configuration Service	Configuration service is responsible to configure onsite park collector modules, single board computer coordinators and multiple parts of the monitoring system. Configuration Service does not provide the capability of updating the modules on runtime.
Discovery Service	The service for discovering all the modules during runtime. A simple ping by this service, will list all the module services with a corresponding health status. For the microcontroller collector modules, the Discovery Service cannot ping, because the microcontroller does not have any API endpoints and does not receives any request. Therefore the microcontroller must constantly feed with data the discovery service.

Metrics DB & Event Logs DB	The metrics database store all the metrics that are gathered from each collector module and the event log database any event logs that are produced from the discovery service.

The components of API layer use the following software and network protocols:

Component	Software / Programming Languages	Communication Protocols
API Monitoring System	Python 2.7 / JavaScript	MAC / IPv4 / TCP / HTTP - 1.1 or 2.0
Configuration Service	Python 2.7 / JavaScript	ZigBee Protocol & MAC / IPv4 / TCP / HTTP - 1.1 or 2.0
Discovery Service	Python 2.7 / JavaScript	ZigBee Protocol & MAC / IPv4 / TCP / HTTP - 1.1 or 2.0

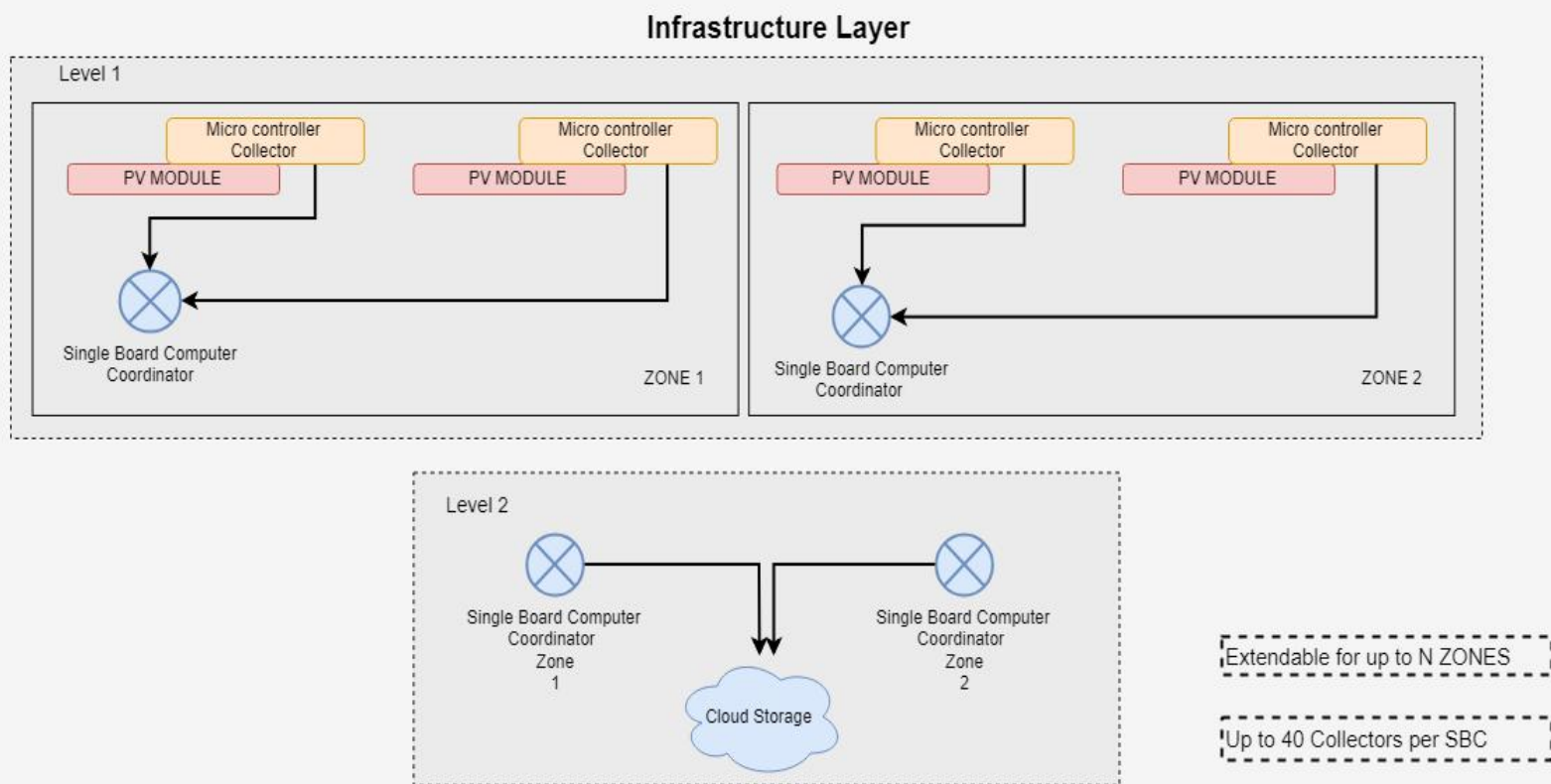


Figure 1.3.4: Infrastructure Layer Diagram that is consisted of Level and Layers

The infrastructure layer is composed by levels and zones. Level 1 has multiple zone networks. Each zone represents a network tree that the root is the single board computer (SBC) coordinator and the microcontroller collectors are leafs. The infrastructure is capable of having as many zones as possible [N zones]. The microcontroller collectors are sending information to the SBCs. Only one Coordinator and one or multiple (up to 40) end devices can exist inside the zone network. The reason is explained on the protocols section and it is because of a limitation of the ZigBee protocol. This literally means a zone is equivalent to a closed network cluster. Level 2 components are the SBC coordinators and the cloud storage. The coordinators that gather data from the collectors, communicate to an external server storage. The limitation on this network model is that it only allows up to forty collectors per SBC. This restriction is applied for several reasons. First for scraping reasons, if for every one minute a scrape happens then all the microcontroller collectors in the network zone must send their data without collisions.

High-speed high-bandwidth isochronous is subject to data loss. So if a SBC has many radio receiving modules on its USB ports, there is a vulnerability to lose information.

Microcontroller Collector

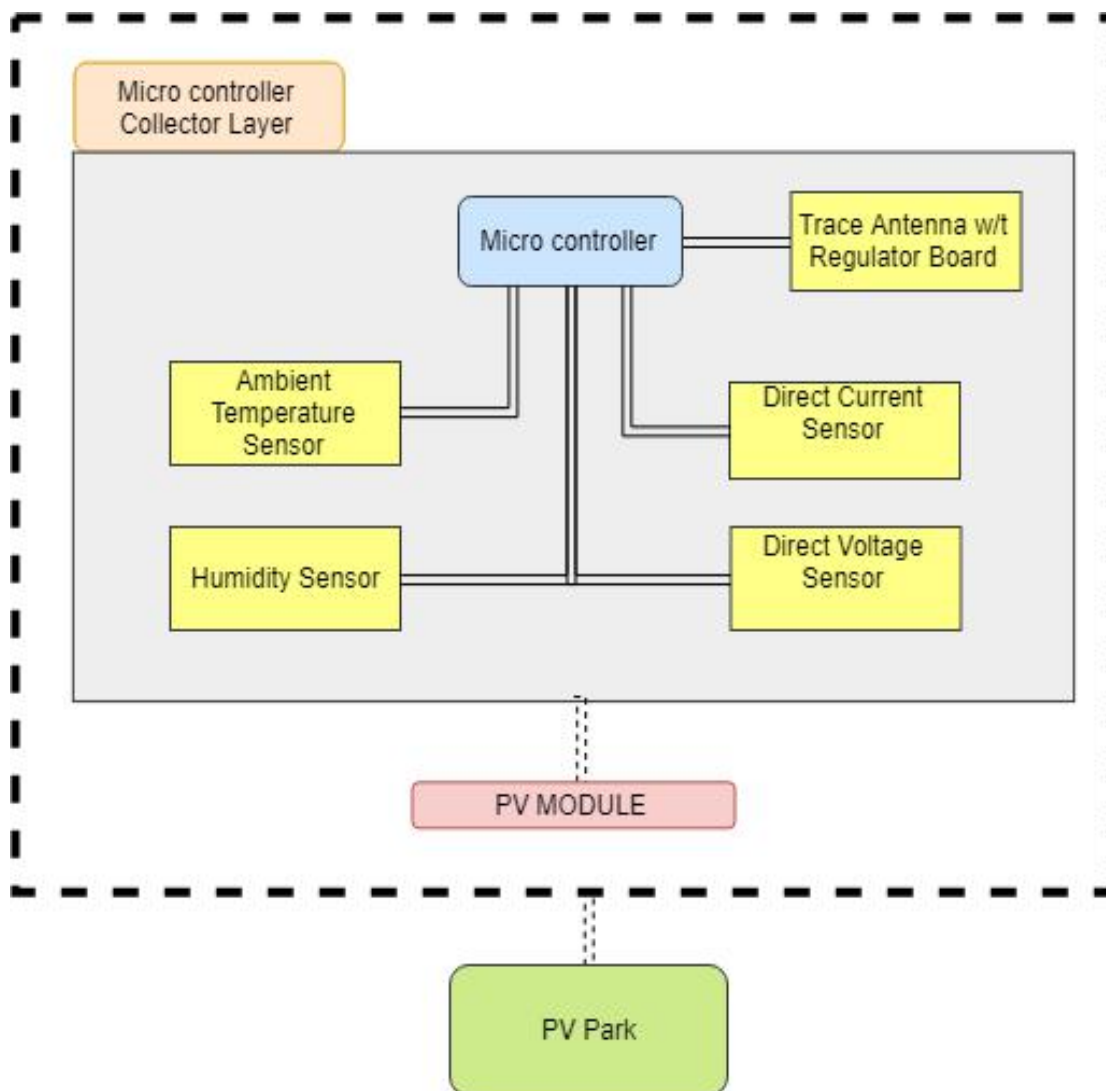


Figure 1.3.5: Microcontroller Collector Layer

The microcontroller has an embedded trace antenna on a regulator board. The prototype is consisted of a breadboard that the sensors rely on. The apparatus has an ambient temperature sensor for measuring the temperature, a humidity sensor for the measuring humidity, a direct current and voltage sensor for measuring current and voltage of the solar PV. A microcontroller does not have any other means for transmitting data except of the trace antenna. A microcontroller typically is a computer on a single monolithic integrated circuit. Microcontroller's program must be stored in the available on-chip memory. Compilers and assemblers are used to convert both high-level and assembly language codes into compressed machine code. Usually, they contain

several to dozens of general purpose input/output pins (GPIO). GPIO pins are software configurable to either an input or an output state. Configured GPIO pins to an input state are often used to read sensors or external signals, in our case the four sensors. Configured to the output state, GPIO pins can drive external devices through outer power electronics such as the trace antenna.

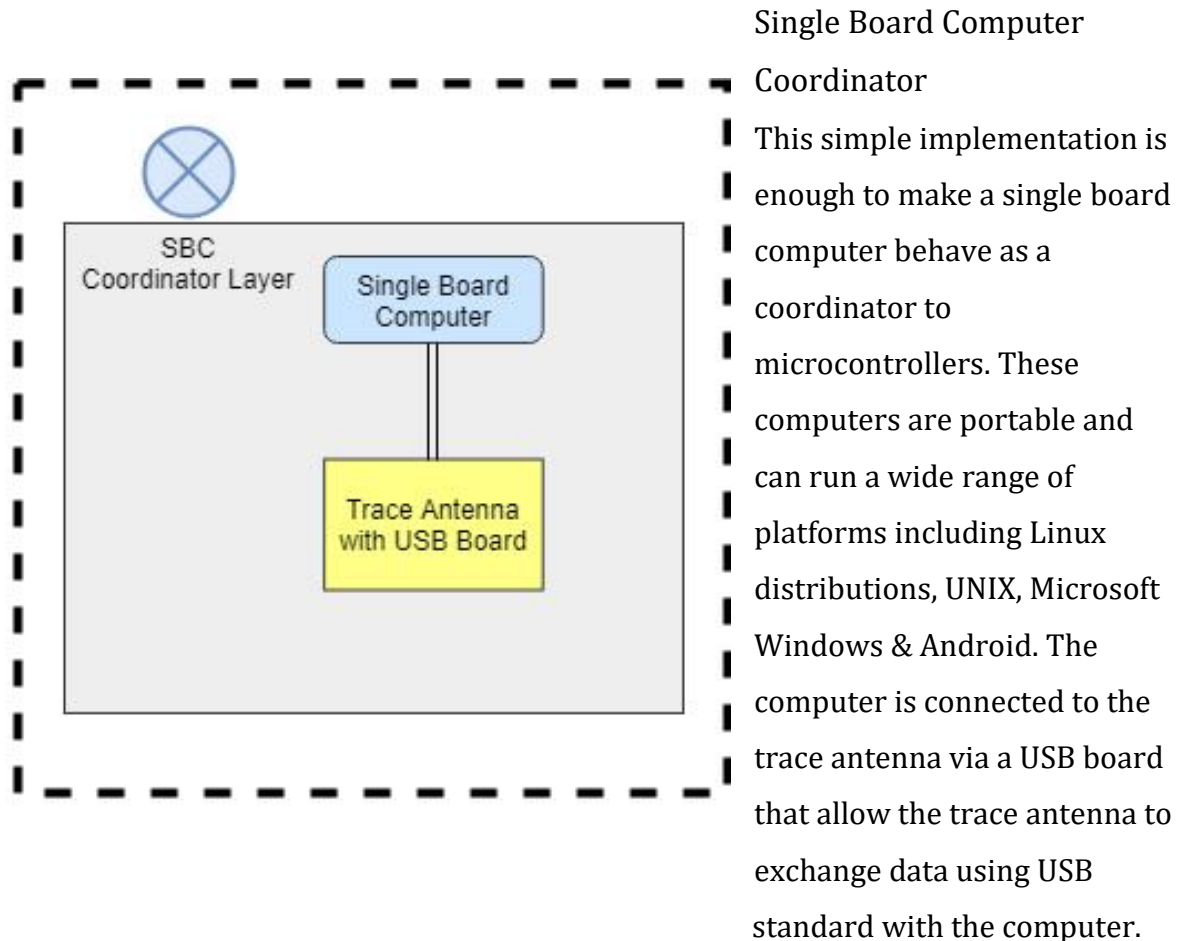
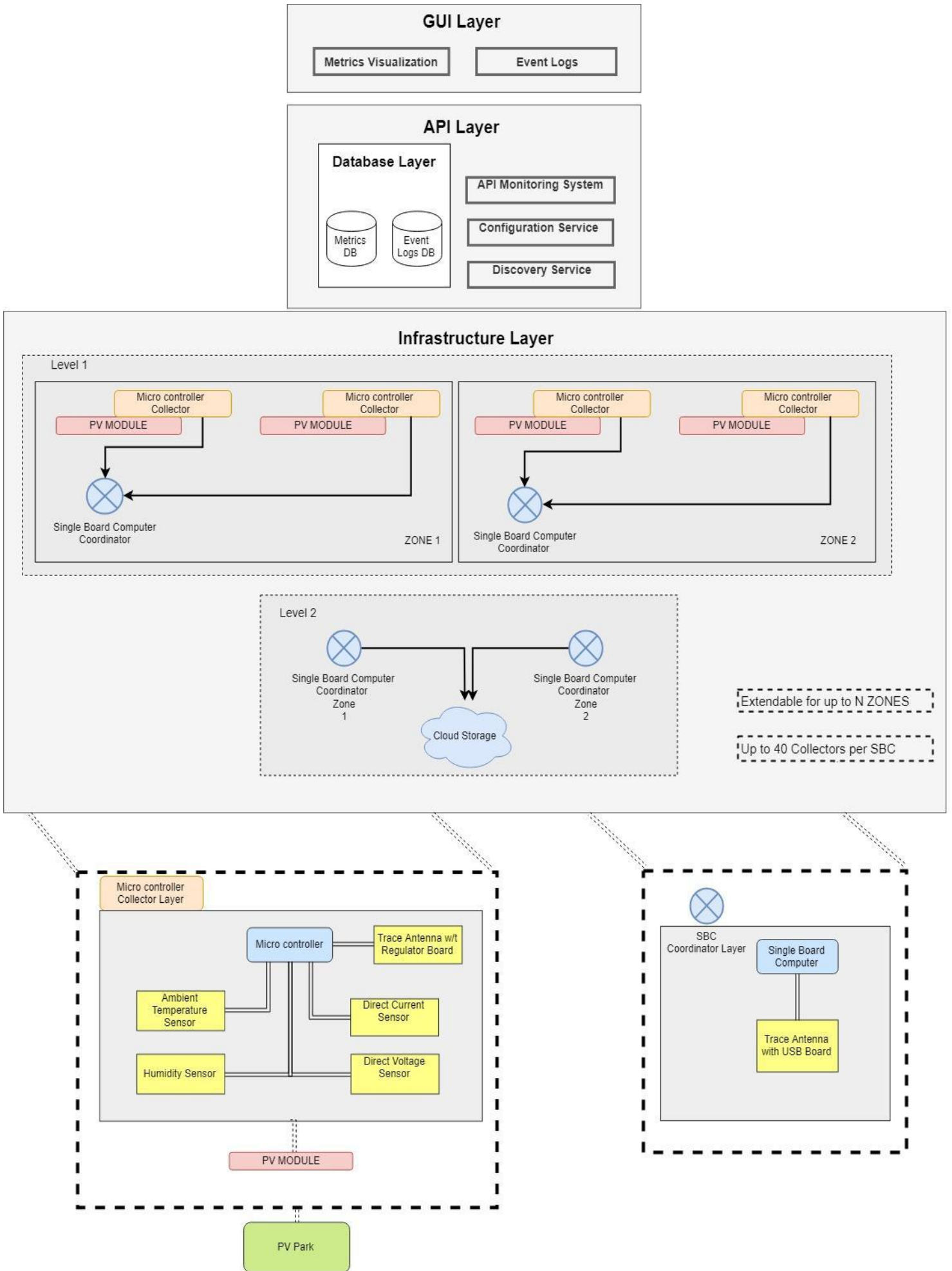


Figure 1.3.6: Single Board

A huge factor for choosing over Computer Coordinator Layer to use a SBC over other modules is the power cost and the computational needs of the problem. “While SBC-based clusters are energy efficient overall, the operation cost to performance ratio can vary based on the workload.”, “The low cost benefit of using SBCs is an attractive opportunity in green computing. These computers are increasingly becoming powerful and may help improve the energy efficiency in data centers.” [Basit Qureshi and Anis Koubaa]

For collecting data from the micro controllers and sending them to a server, a SBC covers the computational needs without much utilization. The time complexity of the problem is $O(n)$, where N is the length of the metrics.

Upon the next chapters there will be an excessive discussion on the implementation of each layer and the chosen products that assemble the components and a justification followed by financial cost of this model for proving our purpose.



Communication Protocols

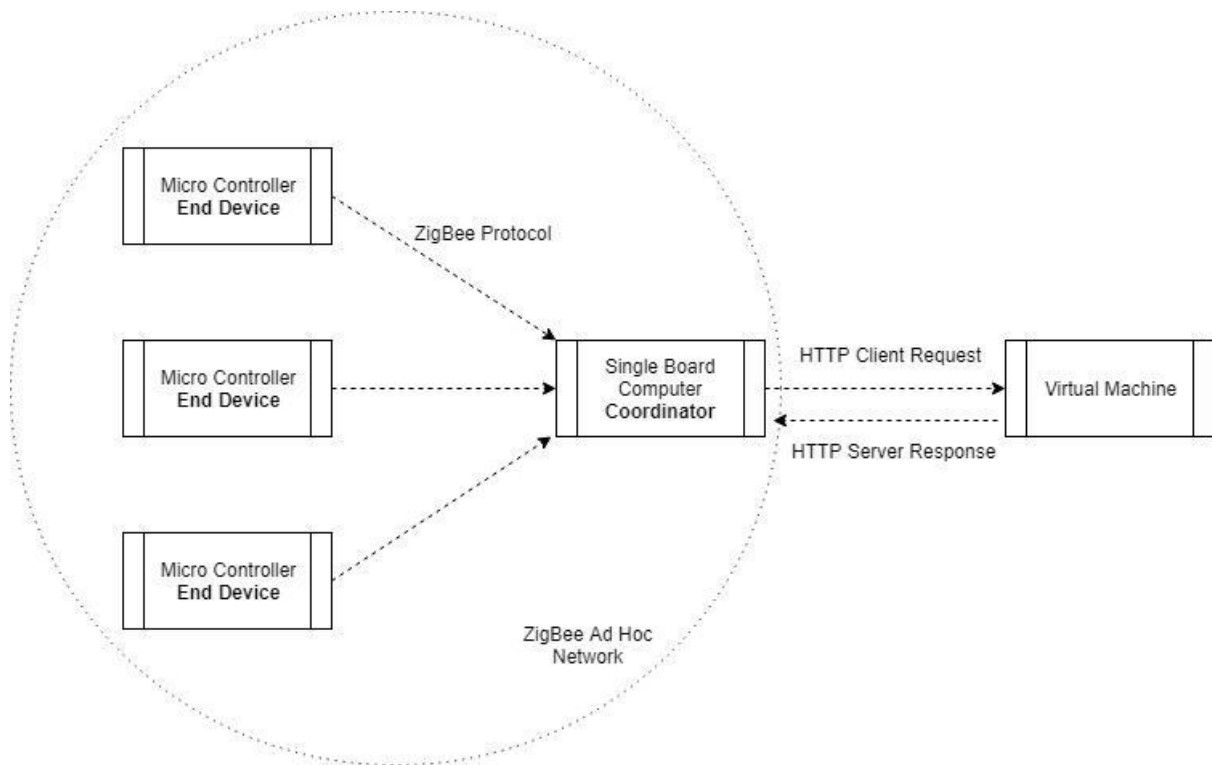


Figure 1.3.7: Network Protocols involved in the infrastructure.

Description of the Communication Protocols

HTTP Protocol

The HTTP is an application protocol and the foundation of the World Wide Web. It is simple, extensible and stateless. The proposed system follows the HTTP communication between client requests and server responses. These messages will be exchanged from the single board computer Coordinator to a virtual machine [figure 1.3.8]. The proposed solution follows transmission control protocol (TCP) for transport layer, internet protocol version 4 (IPv4) for internet layer and medium access control (MAC) for link layer for the network part between the SBC Coordinator and the Virtual Machine.

ZigBee Protocol

ZigBee is a higher latency, lower bandwidth, asynchronous protocol that uses the 802.15.4 standard as a baseline and adds additional routing and networking functionality.

ZigBee can be best described as a mobile ad hoc network because it does not rely on a pre-existing infrastructure, such as routers in wired networks or access points in

managed wireless networks [C. Siva Ram Murthy & B. S. Manoj] but is a self-configuring network of mobile devices connected wirelessly [Morteza M. Zanjireh & Hadi Larijani]. The ZigBee protocol supports three operating modes: a) Coordinator b) Router c) End Device. The Coordinator forms the root of the network tree and might bridge to other networks. There is only one ZigBee Coordinator for a network of end devices and it is capable of: a) establishing a ZigBee network, b) permit other devices to join c) leave the network, d) assign 16-bit network addresses and route network packets. ZigBee Router acts an intermediate router for forwarding data. All the operating modes receive or send network packets, can join or leave the ad hoc network or entering sleep mode [Ankur Tomar, page 15]. End Device main functionality is to keep communicate with the parent node which is a Coordinator node.

Routers and Coordinators maintain a table of all child devices that have joined called the child table which has finite size and determines the number of children that can join the particular Router or Coordinator. The initial release of software on this platform (Digi Xbee) supports up to 40 devices when configured as a coordinator or a router. [Digi Documentation, End Device Capacity]

IEEE 802.15.4

The IEEE 802.15.4 physical layer transmits data using a) a certain radio channels b) spreading techniques c) specific modulation. To implement these techniques it offers these characteristics: multiple channels from 868 to 868.6 MHz, 902 to 928 MHz and 2.4 to 2.4835 GHz. The spreading techniques is Direct Sequence Spectrum which the message signal is used to modulate a bit sequence much shorter than the original message signal also known as Pseudo Noise code which it will greatly increase the occupied frequency bandwidth of the signal energy around the carrier. It's vital to discuss how this protocol ensures data integrity over network collisions. For packet collisions, the IEEE 802.15.4 evaluates the medium activity state with the clear channel assessment (CCA) task. CCA does can be performed in three different ways: a) Energy Direction: If there is a detection of a signal that is above an ED threshold. The threshold based on the bandwidth of the IEEE 802.15.4 channel. b) Carrier Sense Mode (CSM): The detection of a signal with similar spreading characteristics of a signal of IEEE 802.15.4. c) The combination of both a) and b). The CSMA mechanism that ensures to avoid collisions in the network is based on CCA task. Each time the CCA returns a busy channel the CSMA mechanism increases the NB (Number of back offs; failed attempts) else it decreases the Contention Window, which is a variable that represents the

number of back off periods that must be clear before starting transmission. When Contention Window variable reaches 0, the message is transmitted.

No device is assign router mode for this implementation. This network model can advance by adding modules that implement ZigBee routing operation.

1.4 PV Technology & Characteristics

The power output of a solar panel depends on the intensity of solar radiation, the amount of aggregated area of solar cells and the solar cell power efficiency of the modules.

$$E = PV_area * PV_efficiency * Intensity$$

[Page 8, F. Kong and X. Liu]

Solar cell power efficiency refers as the amount of energy in the form of sunlight that can be converted via PV modules into direct current. The proposed system must be able to provide all these parameters in real time to the observer. Other environmental factors such as humidity and temperature are necessary for measuring the performance of the modules. For example power output degrades when ambient temperature is getting higher". According to estimation for every degree rise in temperature, efficiency of PV module decreases 0.5 percent" [Rizwan Arshad, Salman Tariq, Muhammad Umair Niaz, Mohsin Jamil].

Solar irradiance can be measured using an analog to digital pyranometer. Pyranometers are very expensive in the market. A pyranometer that can be integrated on a microcontroller can cost up to \$200 -\$800.[Scientific Campbell]

The following performance parameters are proposed by the ENEDI engineers to be continuously sampled at a 1-minute resolution and accumulated as 15-minute averages for the

number of selected PV modules (i.e. 2 PV modules): 1) Module temperature 2) DC current 3)DC voltage 4)Solar Intensity [FOSS Engineers, page 1]. The proposed system measure all these metrics except solar intensity due to the cost of the pyranometer sensor.

Chapter 2

Microcontroller Collector

2.1 Arduino Uno Rev3

Arduino Uno is a microcontroller board based on the ATMEGA328 next generation microchip. ATMEGA328 is based on an advanced RISC architecture with 32x8 general purpose working registers. The Arduino clock rate is up to 16MHz resonator with a crystal oscillator for dealing with time issues. Inside Arduino there is a self-programmable flash program memory where developers can upload a script that will keep executing without any human intervention. Also it is capable of working between temperatures of - 40°C to 85°C [Arduino Datasheet]. On waking stage at 25°C and 1.8 V as operating voltage, its consumption is 50 mA, but by using power saving techniques such as lowering the voltage, reducing the clock speed and saving power with software the consumption can fall by to up to 13.4mA [Spark fun, page 2]. The Arduino Uno Rev3 board is connected to a computer via USB, where it connects with the Arduino development environment (IDE). The user writes the Arduino code in the IDE, then uploads it to the microcontroller which executes the code, interacting with input and output modules such as sensors. With all these diverse features Arduino is definitely a neat embedded computer platform choice for monitoring purpose projects. The internet community has published thousands of projects that can be done using Arduino.

2.2 Xbee Transmitter module for Microcontroller Collector

Module	Model	Data Rate	Antenna	Power Consumption	Range
Xbee 1mw Trace Antenna	Trace Antenna-Series 1	250kbps	Printed circuit board	50mA @ 3.3v	300 ft

Figure 2.2.1 Xbee Module Table Description

Xbee 1mw Trace Antenna is a radio module that communicates using the protocol 802.15.4 / ZigBee standard protocol. The main mechanism of the module is to perform frequency modulation. The power output of Xbee is about 1 mW and outdoor range is up to 300 ft. Radio frequency data rate can be up to 250 Kbps. Today Xbee is the de facto

solution for internet of things for point to point communication and multi-point networks. [Xbee Datasheet]

Xbee Configuration

Digi International Inc, the company that produces Xbee modules, provides a platform application to enable developers to configure and interact with Xbee modules, the XCTU. XCTU is very simple to use and has friendly GUI. Through the platform, the Xbee transmitter must be configured to act as an end device, as the proposed architecture is indicating. The parameters for the Xbee to behave as an end device and transmit data to a parent node (coordinator SBC) is the channel, Baud Rate and operation mode. Direct addressing requires the sending device to know three kinds of information regarding the receiving device:

However, the Xbee module has to be integrated with a specialized regulator board on the breadboard. The board is populated with 3.3V regulator and carries a FT231X microchip which offers a compact bridge to full handshakes UART interfaces. UART is a common protocol used for duplex serial communication. Xbee modules use UART for serial communication. Sensor values pass through the digital port and then are exposed to a serial port which is used for communication between the Arduino and the Xbee.

Module	Microchip	Function	Compatible
Xbee Explorer Regulated Board	FT231X	Handles 3.3V regulation, signal conditioning, and basic activity indicators	Xbee Series 1/2

Figure 2.2.2 Xbee Regulator Board Table Description [Sparkfun Xbee Documentation]

2.3 Sensors

Sensor	Model	Metric Type	Accuracy (Error Value OR Rate)
Ambient Temperature	DHT22	Celcius (°C)	±2°C
Relative humidity	DHT22	RH (Percentage)	±5%RH
Direct Current	ACS712	Amperes (A)	1.5% at TA = 25°C
Direct Voltage	HCMODU0047	Voltage (V)	1%

Figure 2.3.1 Sensors used with the Microcontroller Collector

[DHT22 Technical Sheet, ACS712 technical sheet, HCMODU0047 technical sheet]

All the sensors listed above demand voltage supply of about 3-5.5V and are connected on a breadboard. For monitoring temperature, DHT22 is selected as it has a very high humidity and temperature measuring accuracy. Measurement range for relative humidity is between 0-100% and -40°C to 125°C for ambient temperature. It uses an 8-bit chip for controlling a polymer humidity capacitor and DA18B20 is used for detecting temperature. Furthermore for monitoring electric current, ACS712 is a fully integrated, low resistance current sensor. The device has a precise linear Hall circuit with a copper conduction path. When current flows through this copper conduction path, a magnetic field is generated which the Hall integrated circuit converts into a proportional voltage. An average sensing period is 2 seconds with full interchangeability. In the market is typically addressed as a very good economical solution for DC sensing using a microcontroller. The HCMODU0047 is simple and very useful sensor which uses a potential divider to reduce any input voltage by a factor of 5. It has the capability to measure up to 25V. This allows the analogue input of a microcontroller to monitor voltages much higher than it is capable of sensing. For example Arduino Uno by itself is only capable of sensing between 0V to 5V. [Arduino Datasheet]. The sensor also includes screw terminals for easy and secure connection of a wire.

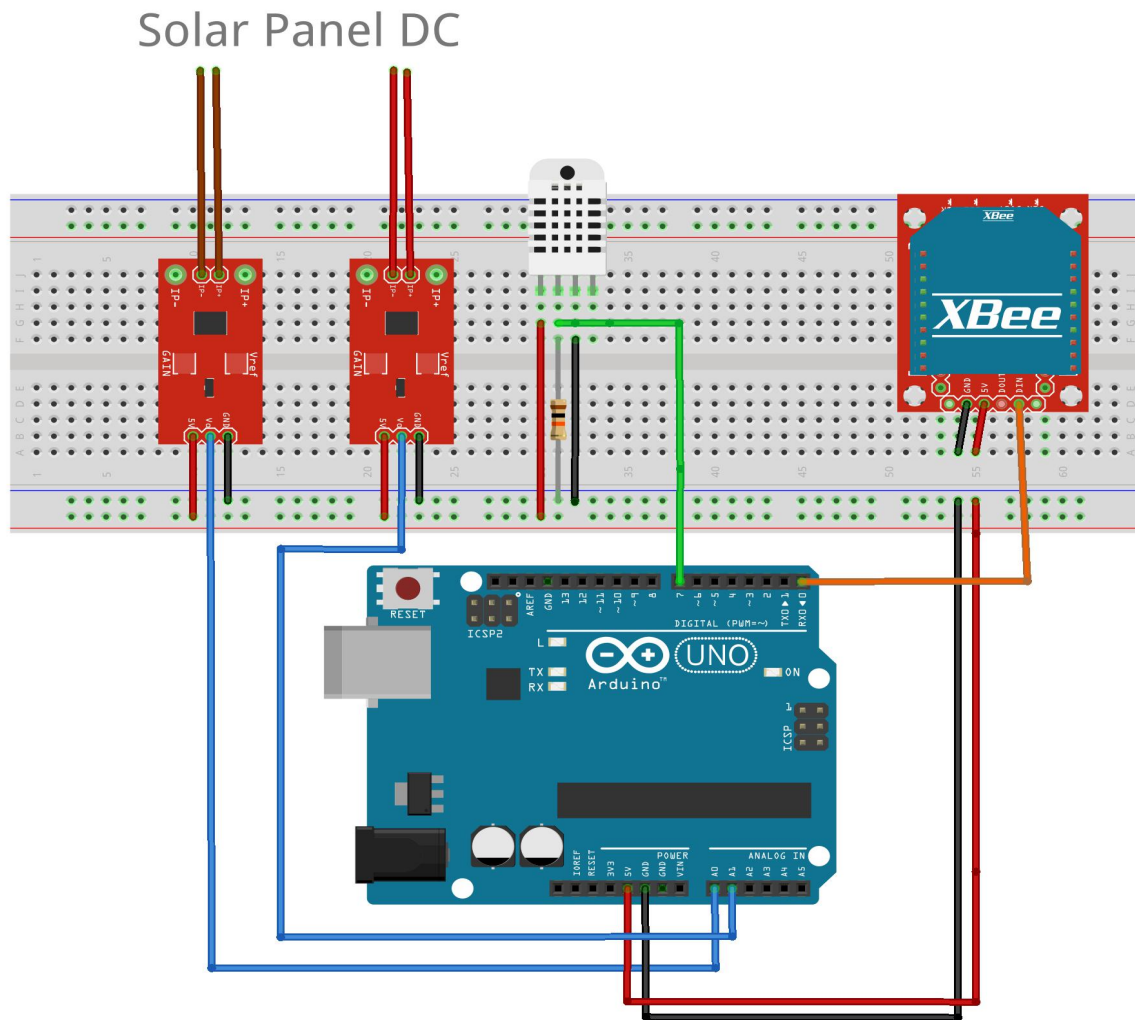


Figure 2.3.2: Proposed Microcontroller Collector apparatus on a breadboard.

Arduino has about 28 pins, 14 digital I/O pins, which 6 provide pulse width modulation output [Arduino Datasheet]. As in figure 2.3.2, the apparatus takes advantage of pin No.7 for digital input values of temperature and humidity and pin No.A0 with pin No.A1 for analog input current and voltage respectively. Pin No.7 is connected with the second data pin of DHT22 which requires a usual 4.7K pull up resistor.

Pin A0 & A1 are an analog pins and are connected with the second analog pins of ACS712 and HCT047 sensors. The pins can read signals from the analog sensors and convert them into a digital value for the microprocessor. Pin 1 TX which is used for serial receiving from the Xbee. When operating, the TX led flashes with baud speed while sending the serial data and RX led flashes during the receiving process. The other two pins that are shown in figure 2.3.2 are used for voltage and ground. Through breadboard as a constructor base, sensors operate at 5V. Each sensor must be connected to the ground and the voltage at the two sides of the breadboard.

2.4 Software on the Microcontroller Collector

Arduino Script

The collector script [collector.ino] is uploaded to Arduino. The script writes the values that gathers from the four sensors to the serial port as human-readable ASCII text.

Each collector has a unique ID from 0 to 4. Remember by the system architecture that every SBC Coordinator has a limit of coordinating forty microcontrollers collectors. For example if we have 3 microcontrollers inside the adhoc network, the IDs are : 0,1,2. The predefined ID has a type of signed integer and is configured as a constant in the script.

Message Structure

The microcontroller creates and exports a message that contains sensor values and is transmitted via the Xbee module. Now the critical part is that the microcontroller collector must send data in a sophisticated manner. A message must be concise and comprehensive for transmitting with less delay and demanding less packet buffer. A possible tuple that can represent a microcontroller message can be indicated as:

XYZ where X is the value, Y is the metric type and Z is the ID of the microcontroller collector.

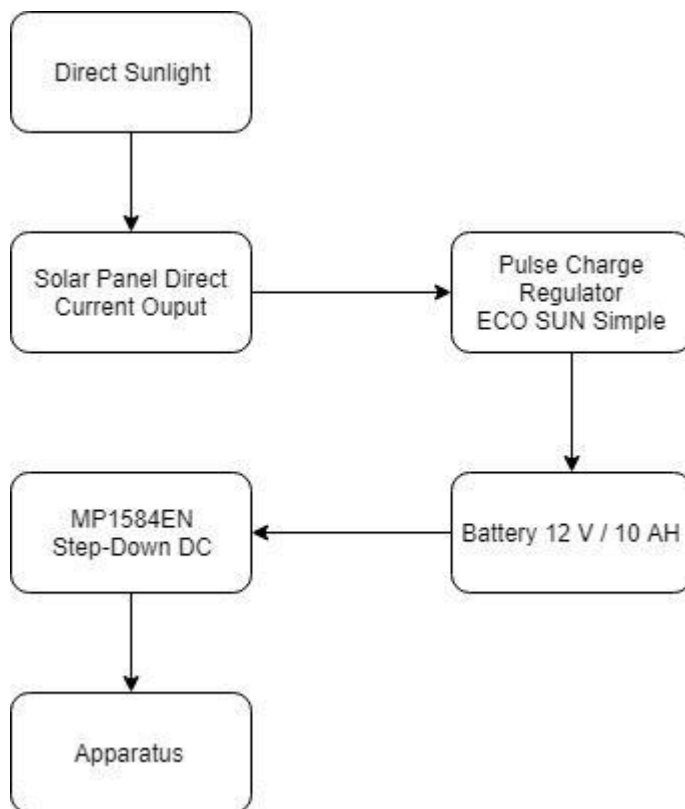
A single message is consisted of 2 integer and 1 characters. That is about 7 bytes that propagate from the Xbee transmitter. The four metrics abbreviations are the follow: T for temperature, H for relative humidity, C for current and V for voltage. The values are integers because there is no interest for precision metrics due to the nature of the problem. Precisions metrics such as floats or doubles need more memory when stored. To sum up, each row represents a message:

Value	Metric	Identification
Integer	C	Short Integer
Integer	H	Short Integer
Integer	T	Short Integer
Integer	V	Short Integer

Figure 2.4.1: Microcontroller message types

For example the message 13T0 means that the microcontroller collector with ID 0 has measured the temperature: 13 °C.

2.5 Power Regulation & Battery



The apparatus will have as main power supply a 12V Battery. Because of the minuscule energy consumption of Arduino on working stage, every microcontroller collector will be powered up from a 12V/22Ah battery outside in the solar park. A typical 12V battery has a capacity about 7 Ah to 10 Ah. That means if we draw 7A with 12V, the battery will last an hour. Arduino needs about 50mA for normal operation. For making sure Arduino will draw correct voltage, a high frequency step down generator must be inserted in the apparatus.

By forcing the current to pass from two input and output wires, the voltage

Figure 2.5.1 Microcontroller Collector Voltage regulation

can be safely converted from 12V to 5V with the module MP1584 [MPS1584 Datasheet]. Another adjustable potentiometer to use is the ECO SUN simple pulse charge regulator 24V to 12V. This is the main regulator that connects the solar panel to the battery. Direct current from the solar panel modules pass through the regulator and is stored inside the battery. The flow of electricity follows the order as shown in figure 2.4.1. Every component of the collector has different voltage demands. This complex regulation might be unpleasant but it is necessary.

Power Cut Scenario

With this model there are multiple possible scenarios to be consider of. For instance direct sunlight might be deficient as a result of an overcast. Suppose that battery is fully charged and contains up to 10A. The sustainability of the collector without direct sunlight is 20 Days:

$$P = I * V$$

$$P_{BATTERY} = 12 V * 10 A = 120 W/H$$

A battery 12V 10A ensures a power output of 120 Watts/Hour. Arduino draws 0.05A at working stage as mentioned before, with 5V operating voltage.

But there is also another component that consumes energy at around 50 mA and that is the Xbee. Therefore Arduino with Xbee module attached can last on the particular battery for about 240 hours or about 10 days.

$$Arduino\ with\ Xbee\ Hours = 0.5/120 = 240\ hours$$

A practical solution is an engineer working for the solar park to recharge the batteries every 10 days. The discovery service can do the following:

1. If those days pass and the Arduino die, the discovery service has to mention the failure of the component.
2. An even better solution is for the discovery service to report the overcast and take responsibility to preserve the power supply to the apparatus.

In conclusion, the theoretical span for the microcontroller collector to have fully functionality is 10 days, after any termination of green power supply.

The discovery service furthermore will commit frequent checks and monitor the supply for the credibility of the power system using other sensors.

Chapter 3

Single Board Computer Coordinator

3.1 Raspberry Pi 3 Model B+

Raspberry Pi 3 Model B+ is a single board computer and functions with the operating system Raspbian. It uses the processor Broadcom BCM2837B0 SoC with a 1.4 GHz 64-bit quad-core ARM Cortex-A53. It supports Ethernet up to a gigabit (1000Mbps, 1000Base-T) and the radio supports 802.11ac WiFi networks running on the 2.4GHz and 5GHz frequency bands, Bluetooth 4.2, and Bluetooth Low Energy (BLE) connections. Raspberry Pi is selected as the core module of the single computer coordinator because it has the capabilities of a mini-computer and covers the computational needs for this assignment. Therefore it demands much greater energy than Arduino Uno R3. Raspberries are required to have a standard alternating-current (AC) electric power supply. [Raspberry Pi Documentation]

Setting the Raspberry Pi

First the RP must be configured with Raspbian operating system. The installation of Raspbian Image must be done on the SD card without the module, a headless install.

1. Format the SD card
2. Download and Extract NOOBS software from raspberry foundation on SD card

After this step, the RP is ready to start. Also the configuration for the Secure Shell is vital for connecting to the RP later. SSH can be enabled by placing a file named ssh, without any extension, onto the boot partition of the SD card from another computer. When the RP boots, it looks for the ssh file. [Raspberry Pi Documentation]

3.2 Xbee Receiver module on SBC Coordinator

Module	Model	Data Rate	Antenna	Power Consumption	Range
Xbee 1mw Trace Antenna	Trace Antenna-Series 1	250kbps	Printed circuit board	50mA @ 3.3v	300 ft

Figure 3.2.1 Xbee Module Table Description

Module	Microchip	Function	Compatible
Xbee Explorer USB	FT231X	USB to serial base unit for the XBee line.	Xbee Series 1/2

Figure 3.2.2 Xbee USB Board Table Description [Sparkfun Xbee Documentation]

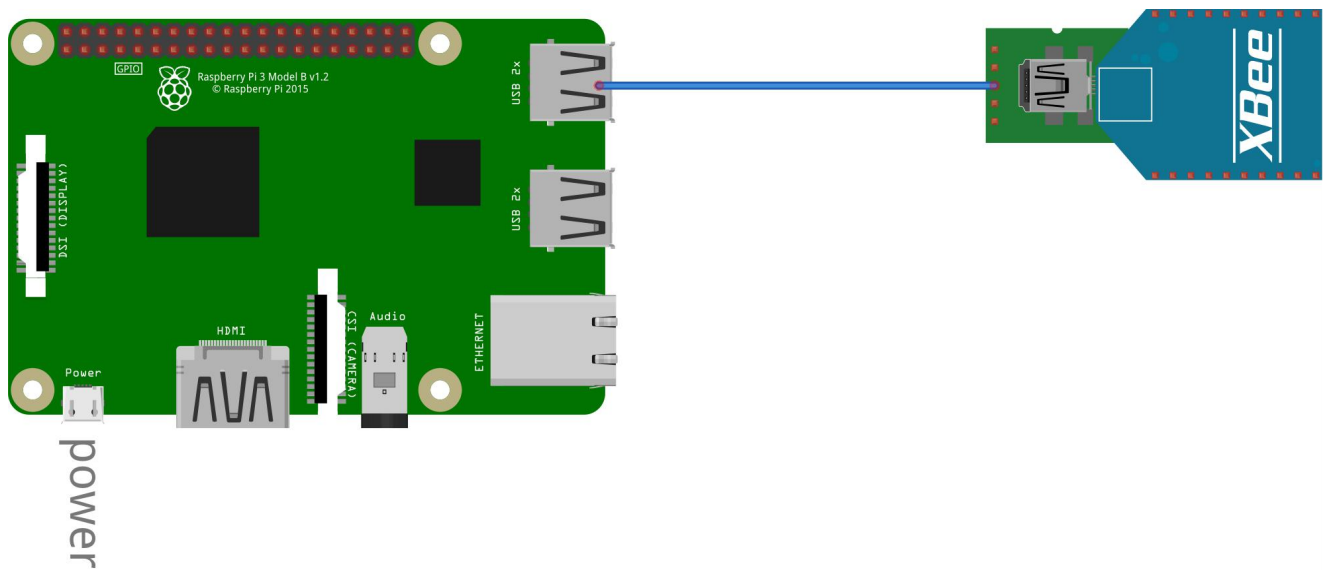


Figure 3.2.3 Single Board Computer Coordinator apparatus

The Coordinator forms the root of the network tree and might bridge to other networks. There is precisely one Zigbee Coordinator in each ad hoc network in level 1 as it is indicated by the system's architecture. The Xbee module is connected to an Xbee USB board. Thus the coordinator has a simple implementation. The raspberry Pi is plugged via a RP compatible power cable to a standard wall power and via a USB cable with the Xbee Explorer USB board via the No.1 USB port. The Xbee Explorer USB board connects the Xbee module to the SBC, allows data exchange and access the serial and programming pins of the Xbee module. Its main embed component is an FT231X USB-to-Serial converter chip that translates data between the computer and the Xbee, same as the Xbee receiver module regulator board.

3.3 Software on the SBC Coordinator

8-N-1 Protocol & Python Library

Before the software implementation is discussed, it is important to mention the 8-N-1 shorthand abbreviation. 8-N-1 Protocol is a physical layer protocol that is used by universal asynchronous receiver-transmitter (UART) hardware such as the Xbee module. "8-None-1" is a common shorthand notation for a serial port parameter setting or configuration in asynchronous mode, in which there are eight (8) data bits, no (N) parity bit, and one (1) stop bit [Faranak Heidarian]. It refers to the standard breakdown of data words in the serial format. As referred before, Xbee modules use UART to communicate with USB and baud rate, which is the serial communication speed between the modules (9600). It is essential to find the correct configuration on how to read bits coming from the Xbee module.

Data gathering from Xbee module

In python library "pyserial", which is a specialized library for reading data from serial ports, there is a command to open a port at "9600,8,N,1" with no timeout which is to follow the 8-N-1 concept and have the baud rate of 9600 bps.

To open a serial port and fetch data from Xbee module using pyserial, a listener is implemented:

- 1.By calling the function `pyserial.Serial()` and setting as a parameter the string `('dev/ttyUSB0')`, the listener is commanded to follow the 8-N-1 concept.
2. Inside a forever loop, a read line command will keep reading data from the serial port where the XBEE is connected.

Coordinator python script [coordinator.py] in the code appendix illustrates this scenario.

Remote Accessing Raspberry

SSH Protocol

Communication with Raspberry Pi can be done via SSH protocol. PUTTY is an excellent SSH and telnet client that provides a graphical user interface for setting any option of the SSH. For accessing the RP via SSH, hostname and port must be given. Hostname is the local IP address of the Raspberry Pi, which is dynamic if not configure static via the DHCP configuration file. The port used for SSH is 22.

Remote Frame Buffer Protocol

Another program used was RealVNC which offers a secure ready to use remote access with graphical desktop sharing (Virtual Network Computing). It uses the Remote FrameBuffer protocol (RFB) to remotely control the targeted Raspberry Pi. For accessing via RealVNC a hostname of the RP and port number (usually port No.1) is needed.

IP Address of SBC

A problem occurs with the IP addresses of the SBC Coordinators. Initially inside the local network, each Coordinator has a local IPv4 address that ranges from 192.168.0.0 to 192.168.255.255. All the SBCs are connected via an Ethernet cable to a router, for getting a connection over the internet. In other words the network is composed of the SBCs and a router. In this case, the default gateway that uses the internet protocol suite that serves as the forwarding host is the router. The only external IP that is known is the router's.

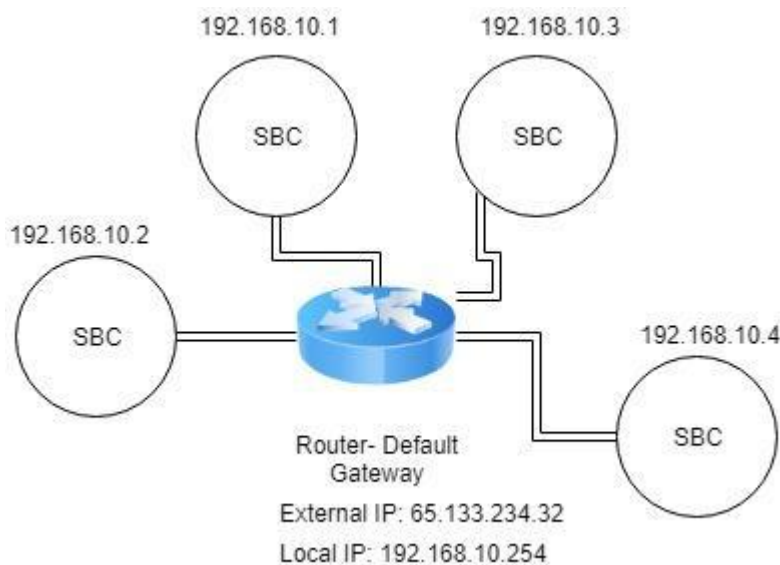


Figure 3.3.1: Example of assigned IP addresses in the network

Port Forwarding

In order to remote access a SBC, port forwarding must be done. Port Forwarding is the process of forwarding data to a node inside a local network with the address of gateway node and the port that much to the target node. Port forwarding can become complicated when there are many instances we want to access with specific ports inside our network. For secure port forwarding Remote.it services were used [Remote IT].

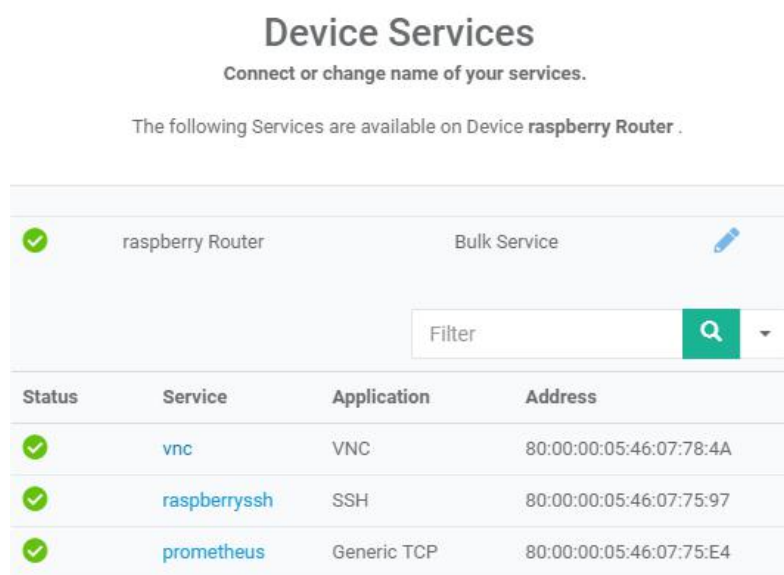
Remote.it assigns an external ip address to the module using secure proxies. For example a SBC is assigned the network address “proxy50.rt3.io.” as an external IP address.

Configuring Remote.It

To install Remote.it services, the SBC must have the weavedconnectd linux package and an account to the Remote.It Company.

For a SBC Coordinator the following services were configured:

1) SSH for remote access via command line, 2) Generic TCP (port 9090) and 3)VNC for remote access with graphical desktop sharing. An example of new address for SSH application was proxy51.rt3.io with port 30805.



The screenshot shows the 'Device Services' interface. At the top, it says 'Connect or change name of your services.' Below that, it states 'The following Services are available on Device raspberry Router .'. There is a search bar with the text 'Filter' and a green search button. Below the search bar is a table with four columns: Status, Service, Application, and Address. The table contains three rows of data, each with a green checkmark in the Status column.

Status	Service	Application	Address
✓	vnc	VNC	80:00:00:05:46:07:78:4A
✓	raspberryssh	SSH	80:00:00:05:46:07:75:97
✓	prometheus	Generic TCP	80:00:00:05:46:07:75:E4

Figure 3.3.2 :Remote.it device services on a SBC

Message Decomposition

In python, for decomposing the message the regular expression, the library “py.re” is used. For example the code line `re.search(r'(\d*)'+_CURRENT,incoming,re.I)` finds the value of current inside an incoming microcontroller message. Each message has three parameters as mentioned before. After decomposing and collecting the three parameters from the message, a request is initiated to the local server of each SBC Coordinator:

Target Url Request : localhost:8000/store_metrics

All these processes are done inside the Coordinator script [cordinator.py].

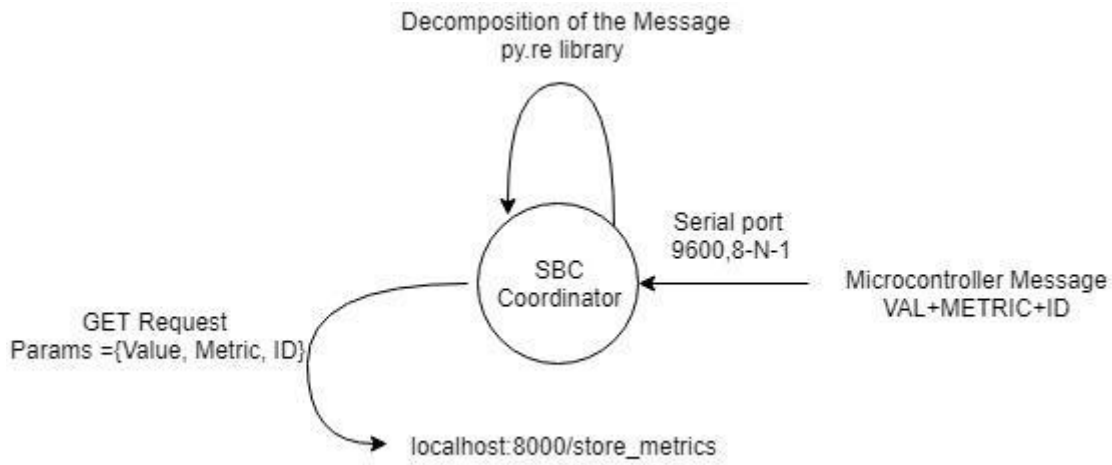


Figure 3.3.3: SBC Coordinator Jobs for storing metrics

Synopsis of the coordinator script module:

1. Open the serial port following 8-N-1 concept.
2. Set up a listener for capturing the incoming microcontroller message.
3. Decompose the microcontroller message to metric, value and ID.
4. Create & send a request to the local server with the above parameters.

Local Server

Pyramid Web Framework is used for implementing the local server on the SBC coordinator [Pyramid Documentation]. To start the server a special server bash script [server.sh] calls the function *serve()* inside the server script [server.py]. Before starting the server a configuration has to take place. A configuration is a special class configurator that the Pyramid Web Framework offers, that has many properties. For the current implementation each server has two accessible paths: 1) `/store_metrics` and 2) `/expose_metrics`. By adding a route to them, each time a request appears, a special function that is configured to that route initiates. Routes are added manually on the properties of the configurator class.

The *store_metrics()* function that routes to `store_metrics` path has to safely store the request parameters from Coordinator script in a MySQL Database. First it extracts the parameters from the request and then another MySQL behaviour script creates an insert query. [databaseHandler.py]

The *expose_metrics()* function that routes to `expose_metrics` path has to fetch from the SBC Coordinator MySQL Database the most recently record of each microcontroller using ID with the four metrics. For example if there are three microcontroller collectors in the MySQL Database a response from the `expose_metrics` function should encode

twelve most recent records, that is, four metrics (T, H, C,V) times three IDs with the latest timestamp.

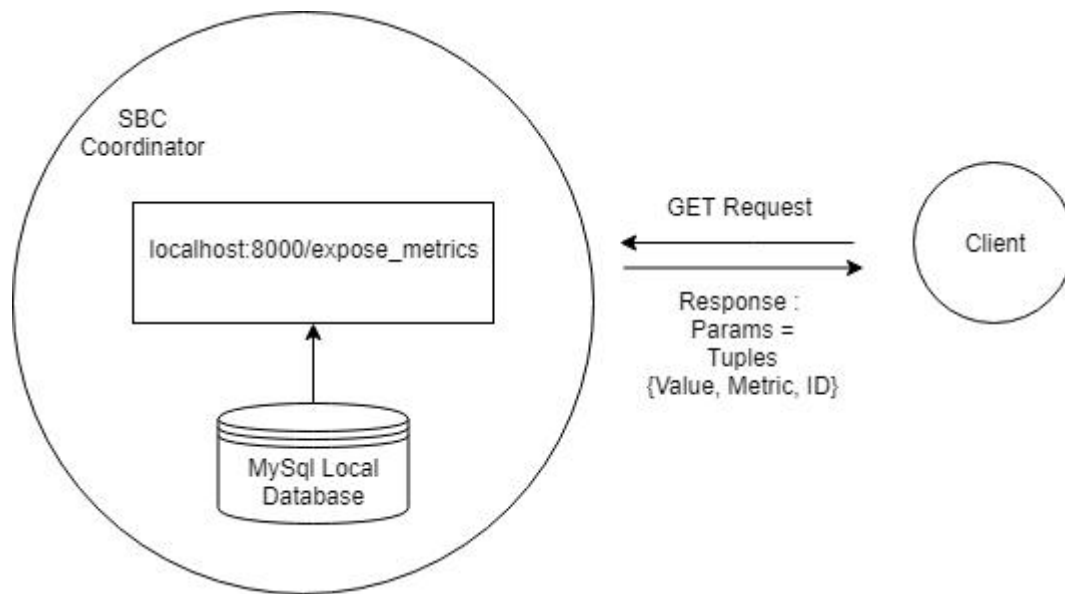


Figure 3.3.4: SBC Coordinator exposing metrics

MYSQL Database

Every coordinator SBC has a simple MySQL database installed on localhost. The database is managed by the database handling script [databaseHandler.py]. This script provides the functionality for insertion, selection and deletion along with a database connector. As mentioned before the server script calls the database handling module for storing data. When insertion happens, the database handling module adds a timestamp with the exact time of the insertion on the tuple. On the other hand, when exposing the values, the selection of the freshest tuples is done by the selector script using a select query. Delete module is a basic MySQL event that triggers every 14 days and delete the metrics according to the oldest timestamp.

Chapter 4

Cloud Virtual Machine

4.1 Virtual Machine Characteristics

The virtual machine (VM) is the final end point of the infrastructure layer. A VM will have the role of cloud storage as indicated by the system architecture. For choosing a VM it is essential to choose the correct amount of CPU cores, RAM, disk space and bandwidth according the computational and memory needs and search for a good company with a reasonable monthly rent. The most important aspect although, is the disk space. All the SBC Coordinators are exposing their metrics to the VM and then, are stored in TSDB format. [TSDB Repository]

Disk Space & API calls

Disk space is affected by the number of PV modules inside the park, the metrics that are gathered from each one and the scrape interval. For example a one day total memory equation would be:

$$Bytes_{day} = Bytes_{Microcontroller\ message} * PVs * ScrapeInterval$$

Suppose a solar park has 200 PV modules and each minute a scrape from the microcontroller collectors takes place. Also as mentioned before, each microcontroller message is 8 bytes and each microcontroller on the PV module has to give 4 metrics: $200 * 7 * 4 * 24 * 60 = 8 \text{ MB per day}$, that is 2,92 GB per year. An average disk space for renting cheap VM is about 25 GB.

API calls must be also calculated for every scenario. For the given scenario there are about 1152000 incoming requests from the SBCs per day or 800 API calls per minute. An average VM with 8GB RAM and a CPU with 4 cores can easily handle that.

4.2 Grafana Framework

For data visualization, a leading open source software for time series analytics is Grafana Framework. It combines a friendly graphical user interface with a powerful dashboard for any database. Workflow integration includes authentication and a variety of themes to choose. It also provides a build in monitoring system support with a query editor integrated with a metric name lookup and template queries for dashboard and discover pattern of series.

Chapter 5

Prometheus Monitoring System

5.1 Prometheus Concepts

Prometheus is a state of the art open source monitoring and alerting toolkit. It provides client libraries that allow easy instrumentation of many services. All the data that the Prometheus gather are stored as time-series.

HTTP Endpoint

A very important concept is that Prometheus collects metrics from monitored targets by scraping metrics HTTP endpoints on these targets. An HTTP endpoint is where the timestamp values are available for scraping. An instance is an HTTP endpoint (hostname and port). Each scrape job can target multiple instances. Jobs and instances are configured inside the Prometheus configuration yml file. [sbc_configuration.yml]

Example of a job with multiple instances:

```
job: api-server
```

```
  instance 1: 230.157.60.4:5670
```

```
  instance 2: 230.157.60.4:5671
```

Prometheus Metric Type

Each value that is collected, is followed by a timestamp. The client library offers a variety of metric types. Particularly since the monitor parameters are temperature, humidity, current and voltage, the chosen metric is gauge, which is a metric that represents a numerical value that can arbitrary go up or down.

5.2 SBC Coordinator Prometheus

Each SBC coordinator has Prometheus installed and listens on localhost:9090. The instance that the SBC is scraping is the local web pyramid server. Inside the yml configuration file, the host and port of the instance are those of the pyramid web server (localhost:8000) and scraping is by default every 15 seconds. Scraping can change if the needs of the center for scraping is much less aggressive (ex. FOSS, 60 seconds). The job of each SBC coordinator has the name coordinator(ID) where ID is the ID of the coordinator. [sbc_configuration.yml]

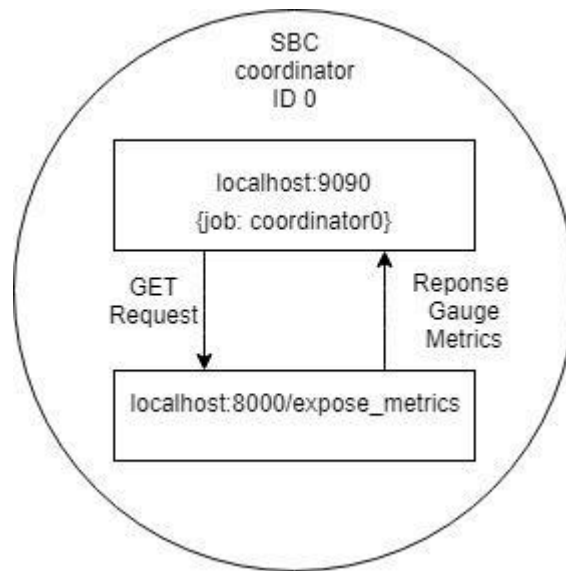


Figure 5.2.1: Local Prometheus interacting with Local Pyramid Server

5.3 Hierarchical Federation Prometheus

Prometheus system offers hierarchical federation which allows a high level Prometheus to scrape data from another Prometheus. Prometheus exposes data in the same manner as HTTP endpoint (localhost:9090/metrics). The Virtual Machine has Prometheus installed and is referred as global. The SBCs Coordinators Prometheus belong to the low levels of hierarchy. The configuration for the global Prometheus is the following:

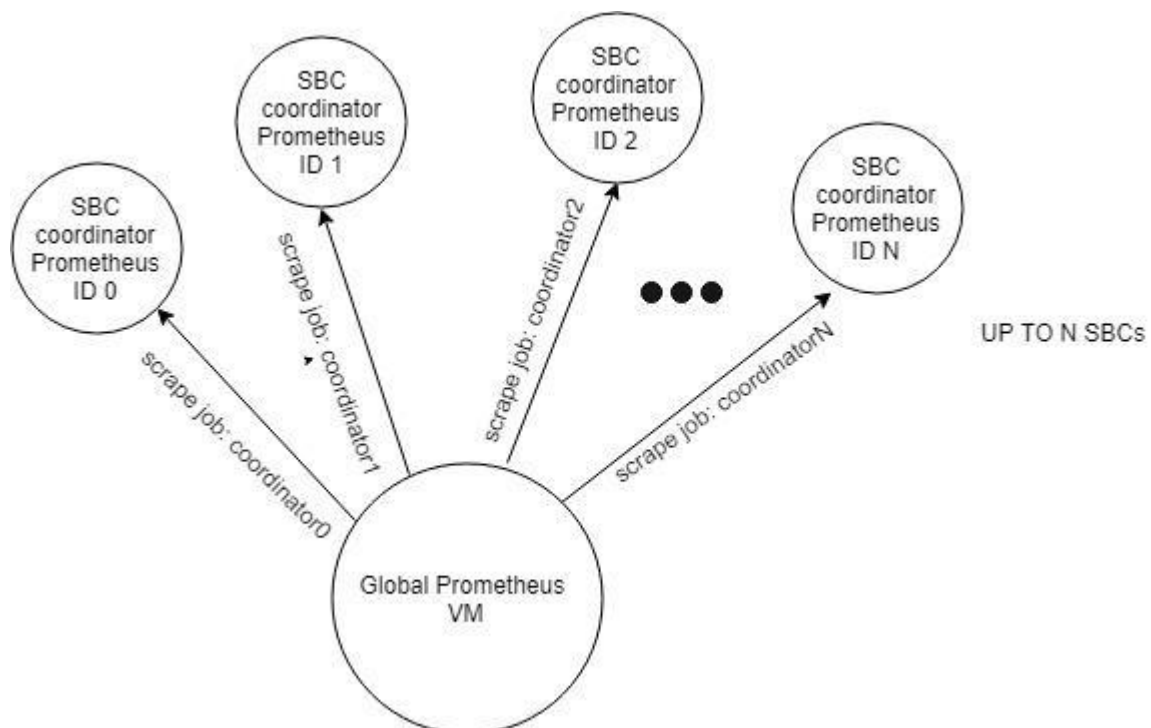


Figure 5.3.1: Global Prometheus scrapes Prometheus Coordinators

5.4 Prometheus Instrumenting

Client Library

In order to monitor services, a Prometheus client library must be chosen. The chosen library is Prometheus Python Client [python client repository]. When instrumenting, there must be a web server that uses the selected library for exposing the Prometheus registry. The Prometheus registry is a set of metrics collectors, in our case gauge metric type. These two steps were implemented in order for the Prometheus to scrape from SBC Coordinator Pyramid server:

- 1) Using Prometheus python library a registry is created with four gauges. (4 Metrics)
- 2) Expose the registry using the Pyramid server for the Prometheus to scrape it.

Instrumentation - Best Practices

When instrumenting we should be sure that is an internal part of code. Each time a new metric appears, a new gauge metric class must be instantiated. A gauge class constructor takes the arguments of the name of the gauge metric, its description and the registry to be assign. Example: *gT = Gauge("Temperature", "Temperature Of Solar Panel 0", registry=registry)*. After that the gauge takes the value of the metric, example *gT.set(temp)*. When implementing the registry collector, it is prohibited to use the usual direct instrumentation approach, which is to set each time the same gauge class with a different value on each scrape. Instead each time we must instantiate a new class gauge and add it to the registry.

This client-server model implementation makes the Prometheus to act as a scraper, not a pusher. Prometheus, every time lapse collects data from a specific endpoint (/expose_metrics). Another different implementation was to set a Prometheus as an endpoint and push json files from the local pyramid web server. Local Prometheus with only one scrape collects the metrics that the local pyramid web server expose with the expose_metric handler.

Chapter 6

Architecture Justification

6.1 Architecture Principles

- a) Maximum benefits at the lowest costs and risks: Due to the modularity of the current architecture there is a maximum benefit at the lowest risk because of the absence of an expensive component.
- b) Compliance with policies of the system running under a solar park business or research center.
- c) Shared Information: The time plots & TSDB databases can be accessed by anyone that has correct API credentials. This is less expensive to have the database to be maintained in a single unit of hardware and multiple backups.
- d) Technology Independence: The API system does not depend on specific technological platform and can function on a different operating system when

To implement these ideas and deliver a real working prototype, there must be enough justification of the hierarchy of this model. The main qualities that the proposed system have, are derived from the architecture hierarchy. These qualities are sustainability, modularity and flexibility.

Sustainability & Modularity

From top to bottom, each node must be interchangeable. This means that if a fault come across, it needs to be changed without dissipate a lot of time and money. Beginning from the metric gathering nodes, the apparatus must be inexpensive and efficient. Moving on, each Coordinator node must be again an inexpensive microcontroller that can handle many requests per second. Because of the absence of many computations, a virtual machine is able to withstand the storage phase. The virtual machine is basically the last level architecture. To be further precise sustainability is basically affected by a lot of factors:

6.2 Architecture Information

Electricity Drawing

As mentioned before, the power supply of raspberry Pi is about 2.5A and has a voltage of 5V. It takes roughly 80 hours of operation to peak the consumption to about 1kw. Assuming that there are 8765 hours in a year the consumption aggregation is 110 KW. The single domestic use tariff of AHK is about 9, 46 cents. Therefore a router node will consume about 10,40 € per year. Arduino however have much lower consumption. 0.05A and voltage 5V it will take about 4000 hours to consume 1 KW. Therefore a node gathering node will consume about 2, 20 € per year. A desktop computer uses about 45-250 W but most laptops about 15W-60W. Let's suppose that a VM might theoretically use 20 W. It might need about 16, 5 € per year.

Data Volume

Data that is generated costs to store. Sophistication on data collecting is shrinking the data to the point only to gather them without any unnecessary text characters, timestamps. Unnecessary junk data are dispatched. When gathering, a set of rules is applied to decrease data volume as mentioned before.

Environmental Protection Actions

To protect the apparatus from the environment, a shield for each node is compulsory. Water from storms, heat waves, wandering animals or even a misdoer can damage the machines. A basic shield can be 120 * 120 cm box made from wood and metal, integrated with a locker.

Interchangeability

Each component that is found on the node can be cheaply purchased in case of any malfunction. The most costly component is the Raspberry Pi, which costs about 40\$. The time needed to change any component is very slight because all the sensors and modules are not embedded.

Chapter 7

Optimization

7.1 Alternative Infrastructure Architecture

There is a limitation on scraping, only 40 microcontroller collectors are allowed to communicate with a coordinator as we have seen before. The current implementation is similar to a star topology. This different architecture is proposed as a more economical solution for many factors:

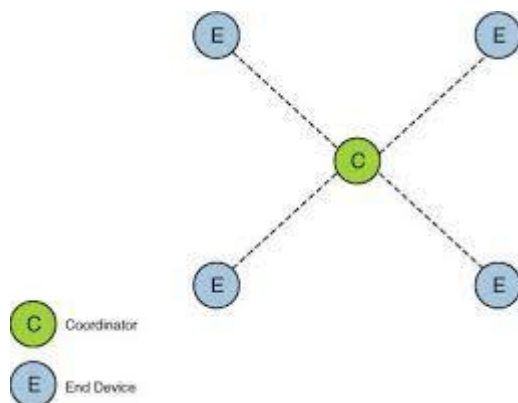


Figure 7.1.1: ZigBee Star Topology

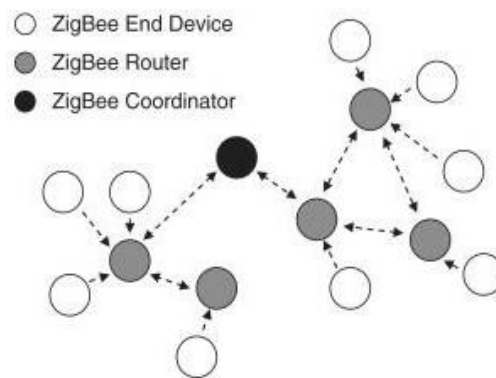


Figure 7.1.2: ZigBee Mesh Topology

The communication in a ZigBee network following star topology is simple. Every collector (end device) is sending data to unique Coordinator, a root node. Thus star topology follows a centralized controlling approach with one node controlling others. This might be adequate for several reasons 1) the sensor selected in as ZC will drain out its battery resources in a fast pace. The second reason is that the IEEE 802.15.4 ZigBee cluster has a limited addressing. That means on a large scale WSN there will be a scalability problem. Resolving this problem is essential and the only way is to follow a different network topology. By following a different network topology means that some ZigBee modules must have router operation inside our ad hoc network.

Mesh topology is a better alternative solution as it enables better scalability and network flexibility but adds a routing protocol overhead. This model is way more power efficient for the Coordinator since the communication rely on many nodes. In order for the protocol to be functional, a ZigBee Router must participate by sending multi hop routing messages in the mesh network and associate with ZigBee Coordinator or another ZigBee router.

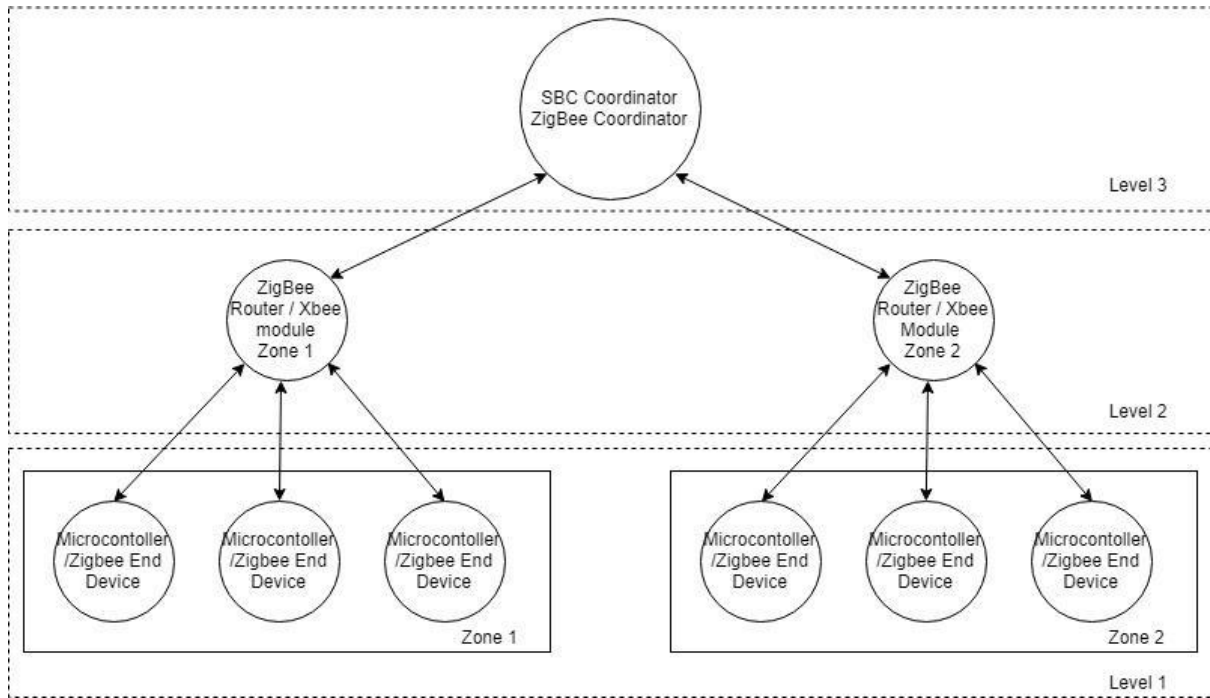


Figure 7.1.3 : Proposed Network Topology

Scaling for Xbee communication

Suppose there are N Solar Panels inside a clean energy park. For every solar panel there is a demand for an onsite gather node, an end-point. The number of routers/coordinators is N & M , where $N > M$. A big issue is to find the optimum number for which one coordinator will handle. The objective is to minimize errors, report without delays failing modules and most importantly create a sustainable time schedule for the propagation of information so that all modules acquiesce with each other. A good scraping configuration and an observer for reporting the failing modules are the keys for implementing the previous monitoring structure on large scale basis.

The Xbee modules are defined by IEEE 802.15.4 standard which specifies carrier sense multiple access with collision avoidance. So the modules have already a build in mechanism that avoid network collision.

Scheduling Microcontrollers

For successful scheduling, a very important value to set is a timeout threshold. A timeout threshold defines the length of time where all the end nodes transmit their data to the selected router.

Therefore the time threshold is mathematically defined as follows:

$$t_{length} = \sum_{i=0}^n x_i + c \quad i = \{0, \dots, x\}$$

X denotes each end device delay that sends the information for the specific router and C is add up idle time for spacing the propagations. Endpoint delay is consisted of processing delay, transmission delay, queuing delay and propagation delay.

7.2 Possible Add-ons

Discovery Service on Microcontrollers

ZigBee Parent Nodes (Coordinator or Routers) receives poll request messages from the end device. When it gets the poll request it check its packet queue to see if there are packet data buffered for the end device. After that it sends a MAC acknowledgement back to the end device. If an end device receives the acknowledgment and finds that the parent has no data for it, the end device can return to idle or sleep. Therefore this polling mechanism enables the end node to remain on sleep mode and conceive battery life. A simple discovery service to check the communication between the Xbee is to open a serial connection to the Coordinator Xbee and send the following commands:

send +++

send ATID 1...N (for each ID of the Xbee)

By sending the message “+++” into the serial port, an acknowledgement from the coordinator module is received. For normal operation there must be an “OK” text response from the Coordinator. After that, each Xbee that has paths that through the Coordinator is sensed by the AT command : ATID (ID). Also the NC command can help us determine how many additional end devices can join the router or coordinator.

Microcontroller Power Optimization

A policy on the working time schedule of the microcontroller must be applied. There is no solar energy output from the PVs during night. So at night Arduino electricity consumption must be less than 50 mA. AtMega provides a variety of power profiles and sleep modes such as: 1) Idle, 2) ADC Noise Reduction, 3) Power Down, 4) Power-save, 5) Standby and 6) Extended Standby.

Sleep Mode	Active Clock Domains					Oscillators		Wake-up Sources							Software BOD Disable
	clk _{CPU}	clk _{FLASH}	clk _{IO}	clk _{ADC}	clk _{ASY}	Main Clock Source Enabled	Timer Oscillator Enabled	INT 1, INT0 and Pin Change	TWI Address Match	Timer2	SPM/EEPROM Ready	ADC	WDT	Other/O	
Idle			X	X	X	X	X ⁽²⁾	X	X	X	X	X	X	X	
ADC Noise Reduction				X	X	X	X ⁽²⁾	X ⁽³⁾	X	X ⁽²⁾	X	X	X		
Power-down								X ⁽³⁾	X				X		X
Power-save					X		X ⁽²⁾	X ⁽³⁾	X	X			X		X
Standby ⁽¹⁾						X		X ⁽³⁾	X				X		X
Extended Standby					X ⁽²⁾	X	X ⁽²⁾	X ⁽³⁾	X	X			X		X

Figure 7.2.1: AtMega Modules in Different Sleep Modes.

Every mode has different active clock domains, oscillators and wake up resources. As one can see, putting microcontroller into power down state is the lowest current consumption state. Since there is no demand for the microcontroller to gather or output anything, the power-down sleep mode is the optimum to apply. To set this mode, there are specific registers that must be written to logic one or zero.

Sleep mode Control Register contains 4 control bits for power management [AtMega Datasheet, page 39]. For power-down sleep mode bits : SM2, SM1, SM0 must have the corresponding values: 0 1 1. A lightweight low power library for Arduino that configures these modes by just calling some functions is freely provided by the Rocket Stream Electronics company. [Low Power Library]

Photoresistor to Predict Overcast

A photoresistor is a light controlled variable resistor that decreases with the increasing incident light intensity. Connecting the photoresistor through the analog ports of Arduino, can easily sense the output values of the resistor and if it is below a threshold, a signal can be transmitted on the discovery service for an incoming overcast.

References

1. Δρ. Νικόλας Λουλούδης (2018) *Σχεδιασμός Συστήματος Εξυπνης Διαχείρισης Φορτίων Κέντρων Δεδομένων*, 4.0 edn.
2. Radwell International (2019) *DataLogger CR3000* , Available at <https://www.radwell.co.uk/en-GB/Search/?q=CR3000+> (Accessed: May 2019).
3. C. Siva Ram Murthy and B. S. Manoj (2004) *Ad hoc Wireless Networks: Architectures and Protocols*: Pearson Education, Inc.
4. Morteza M. Zanjireh and Hadi Larijani (May 2015) *A Survey on Centralised and Distributed Clustering Routing Algorithms for WSNs*: IEEE
5. Ankur Tomar (July 2011) *Introduction to Zigbee Technology* , Volume 1(), Global Technology Centre
6. Raspberry Pi (Trading) Ltd (January 2019) *Documentation Raspberry Pi 3+* ,Available at <https://www.raspberrypi.org/documentation/hardware/raspberrypi> (Accessed: May 2019).
7. Basit Qureshi and Anis Koubaa (January 2019) *On Energy Efficiency and Performance Evaluation of Single Board Computer Based Clusters: A Hadoop Case Study*, 4 February 2019: Electronics 2019.
8. Rizwan Arshad, Salman Tariq ,Muhammad Umair Niaz, Mohsin Jamil (April 2014) 'Improvement in Solar Panel Efficiency Using Solar Concentration by Simple Mirrors and by Cooling', *IEEE*, pp. 292-295.
9. SparkFun Company (2019) Reducing Arduino Power Consumption, Available at: <https://learn.sparkfun.com/tutorials/reducing-arduino-power-consumption/all> (Accessed: May 2019).
10. Allegro, Microsystems Inc (n.d.) *ACS712 Fully Integrated, Hall Effect-Based Linear Current Sensor with 2.1 kVRMS Voltage Isolation and a Low-Resistance Current Conductor*
11. Aosong Electronics Co.,Ltd (n.d.) *Digital-output relative humidity & temperature sensor/module DHT22*
12. EKT, Ltd (n.d.) *Arduino Voltage Sensor Module , HCMODU0047*, Technical Sheet
13. Arduino ,RS, Radiospares, Radionics (n.d.) *A000066 / Arduino / datasheet-38879526.*, Datasheet.
14. Digi International Inc. (n.d.) *XBee®/XBee-PRO® RF Modules*, : Product Manual v1.xEx - 802.15.4 Protocol.
15. MPS (2011) *MP1584 ,3A, 1.5MHz, 28V Step-Down Converter*
16. Faranak Heidarian (2011) *Modeling 8N1 Protocol with Uppaal*.

17. SparkFun (2019) *Xbee Guide*, Available at:
<https://learn.sparkfun.com/tutorials/exploring-xbees-and-xctu/all> (Accessed: May 2019).
18. Pyramid Web Framework (2019) *Support and Development*, Available at:
<https://docs.pylonsproject.org/projects/pyramid/en/latest/#support-and-development> (Accessed: May 2019).
19. Prometheus (2019) *TSDB Format Repository*, Available at:
<https://github.com/prometheus/tsdb/blob/master/docs/format/README.md> (Accessed: May 2019).
20. FOSS Engineers (n.d.) *Proposed Design of Data Logging*.
21. Prometheus Engineers, *Prometheus Python Client*, and Available at:
https://github.com/prometheus/client_python (Accessed: May 2019).
22. Fanxin Kong and Xue Liu. 2014. A Survey on Green-Energy-Aware Power Management for Datacenters. *ACM Comput. Surv.* 47, 2, Article 30 (November 2014), 38 pages. DOI:
<https://doi.org/10.1145/2642708>
23. Jean-Philippe Vasseur, Adam Dunkels, (2010) 'Chapter 19 - Non-IP Smart Object Technologies', in (ed.) *Interconnecting Smart Objects with IP*, pp. Pages 295-302.
24. RocketScream. Lightweight low power library for Arduino.
<https://github.com/rockscream/Low-Power> (accessed May 2019).
25. Campbell Scientific () CS301 pyranometer, Available at: <https://www.campbellsci.com/> (Accessed: May 2019).
26. Digi Company () *End Device Capacity*, Available at: https://www.digi.com/resources/documentation/Digidocs/90002002/Content/Reference/r_zb_en_d_device_capacity.htm?TocPath=zigbee%20networks%7CEnd%20device%20operation%7C__3 (Accessed: May 2019).

The four scientific papers that guided me with the architecture:

1. Sheikh Ferdoush, Xinrong Li, Wireless Sensor Network System Design Using Raspberry Pi and Arduino for Environmental Monitoring Applications, *Procedia Computer Science*, Volume 34, 2014, Pages 103-110, ISSN 1877-0509, <https://doi.org/10.1016/j.procs.2014.07.059>.
2. Gaurav Jadhav, Kunal Jadhav, Kavita Nadlamani (Apr -2016) 'Environment Monitoring System using Raspberry-Pi', *International Research Journal of Engineering and Technology*, 03(04).
3. Fanxin Kong and Xue Liu. 2014. A Survey on Green-Energy-Aware Power Management for Datacenters. *ACM Comput. Surv.* 47, 2, Article 30 (November 2014), 38 pages. DOI:
<https://doi.org/10.1145/2642708>
4. Lizhe Wang, Samee U. Khan (March 2013) 'The Journal of Supercomputing', *Review of performance metrics for green data centers: a taxonomy study*, 63(3), pp. 639-656.

Appendix

1. Definitions

IOT:	Abbreviation for internet of things.
Baud Rate:	Serial communication speed between modules
Serial Bus:	Serial bus is consisted of a transmitter wire and the receiver Wire (RX & TX)
Hall Effect:	The production of a potential difference across an electrical conductor when a magnetic field is applied in a direction perpendicular to that of the flow of current
PWM:	Pulse Width Modulation
Port forwarding:	Forwarding data to a node inside a local network with the address of gateway node and the port that much to the target node
SBC:	Single Board Computer
Microcontroller:	Control device which incorporates a microprocessor.

2. Financial Analysis

Cost of microcontroller:

Cost of SBC coordinator:

Component	Average Selling Price [SparkFun]	Component	Average Selling Price [SparkFun]
		Raspberry Pi 3 B+	\$39.5
Arduino Uno REV 3	\$22.95		
Xbee Trace Antenna Module	\$22.95	Xbee Trace Antenna Module	\$22.95
Xbee Explorer Regulated Board	\$10.95	Xbee Explorer USB Board	\$25.95
Sensor DHT22	\$9.95	Total Amount:	88.40 US\$
Sensor ACS712	\$3.95		
Sensor HCMODU0047	\$3.95		
Total Amount:	51.75 US\$		

3. Scripts

server.sh

```
# Call the serve function of python server
#!/bin/bash
kill -9 $(lsof -t -i:8000) ;python -c 'import server; server.serve()'
```

collector.py

```
# This is the collector module as described in the ADE
# This module collects the message coming from the microcontroller
# decompose it and creates a request to the local pyramid web server.
import time
import datetime
import server
import request
import serial
import re
import time

from prometheus_client import generate_latest, CONTENT_TYPE_LATEST
from prometheus_client import CollectorRegistry, Gauge

#These are the metric characters
_CURRENT = "C"
_TEMPERATURE = "T"
_HUMIDITY = "H"
_VOLTAGE = "V"

#Enable USB Communication
ser = serial.Serial('/dev/ttyUSB0', 9600,timeout=.5);

#Keep reading from the serial port
while True:
    incoming = ser.readline().strip()
    searchID = re.search(r'ID(\d*)',incoming,re.I)
    searchC = re.search(r'(\d*)'+_CURRENT,incoming,re.I)
    searchT = re.search(r'(\d*)'+_TEMPERATURE,incoming,re.I)
    searchH = re.search(r'(\d*)'+_HUMIDITY,incoming,re.I)
    searchV = re.search(r'(\d*)'+_VOLTAGE,incoming,re.I)

    #If its a valid message and holds an ID
    if searchID is not None:
        ID = int(str(searchID.group(1)))
```

```

        print incoming
    if searchT is not None:
        request.createRequest(int(str(searchT.group(1))),_TEMPERATURE, ID)
    elif searchH is not None:
        request.createRequest(int(str(searchH.group(1))),_HUMIDITY, ID)
    elif searchC is not None:
        request.createRequest(int(str(searchC.group(1))),_CURRENT, ID)
    elif searchV is not None:
        request.createRequest(int(str(searchV.group(1))),_VOLTAGE, ID)
    time.sleep(3) # Set time sleep for scraping

```

database.py

```

#Create the connection inside the database

import mysql.connector
from mysql.connector import Error
from mysql.connector import errorcode

_DatabaseName = # Enter Database Name
_UserName = # Enter username database
_Password = # Enter password Database
_host = #Enter host
def createConn():
    connection = mysql.connector.connect(host = _host,
        database=_DatabaseName,
        user=_UserName,
        password=_PassWord)
    return connection

```

MySql Create Statements

Create the four tables

This instructions must be inserted inside MySql:

```

CREATE TABLE IF NOT EXISTS temperature( ID SMALLINT,value INT,time TIMESTAMP);
CREATE TABLE IF NOT EXISTS voltage( ID SMALLINT,value INT,time TIMESTAMP);
CREATE TABLE IF NOT EXISTS current( ID SMALLINT,value INT,time TIMESTAMP);
CREATE TABLE IF NOT EXISTS humidity( ID SMALLINT,value INT,time TIMESTAMP);

```

MySql Insert Statements

#This is the insert python MySql where the values are push to the MySql.

#Called from the python server.

```

import database
import time
import datetime

def insert(value, metric, id):
    conn = database.createConn()
    cur = conn.cursor()
    ts = time.time()

    timestamp = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')

    if metric == 'C':
        sql = "INSERT INTO current (ID,value,time) VALUES (%s, %s, %s)"
    elif metric == 'H':
        sql = "INSERT INTO humidity (ID,value,time) VALUES (%s, %s, %s)"
    elif metric == 'T':
        sql = "INSERT INTO temperature (ID,value,time) VALUES (%s, %s, %s)"
    elif metric == 'V':
        sql = "INSERT INTO voltage (ID,value,time) VALUES (%s, %s, %s)"

    value = (id,value,timestamp)
    cur.execute(sql,value)
    conn.commit()

    print(cur.rowcount, "record inserted.")

```

microcollector.ino (Arduino Script)

```

//This code is uploaded to the Arduino
//You have to also import the Adafruit Sensor Library & DHT
//Arduino Environment ->Manage Libraries
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <DHT_U.h>
#define DHTTYPE      DHT11    // DHT 11
#define DHTPIN       4        // Pin which is connected to the DHT sensor.
#define ID           0        //ID of current arduino this script will be uploaded
DHT_Unified dht(DHTPIN, DHTTYPE);
void setup() {
    //Dht Sensor Configuraton
    dht.begin();
    sensor_t sensor;
    dht.temperature().getSensor(&sensor);
    Serial.begin(9600);
}

void loop() {
    String srlID = String(ID);

```

```

//Voltage sensor
int voltage = analogRead(1);
float temp = voltage/4.092;
float voltage2=(temp/10);
voltage2 = voltage2*10;
Serial.print(voltage2,4);
Serial.println("VID"+ srtID);

//Current Sensor
int RawValue = analogRead(A0);
double Voltage = (RawValue / 1024.0) * 5000; // Becomes mV
double Amps = ((Voltage - 2500) / 66);
Serial.print(Amps,0);
Serial.println("CID"+srtID);

//Temperature Sensor
sensors_event_t event;
dht.temperature().getEvent(&event);
Serial.print(event.temperature,0);
Serial.println("TID"+srtID);

//Humidity Sensor
dht.humidity().getEvent(&event);
Serial.print(event.relative_humidity,0);
Serial.println("HID"+srtID);
delay(2000); // Delays 2 seconds, as the DHT22 sampling rate is 0.5Hz
}

```

request.py

```

#This is for sending a request to the local Pyramid web Server
import requests
__URL= #"http://localhost:80/store_metrics"
def createRequest(value,metric,ID):
    _PARAMS = {'value':value,'metric':metric,'ID':ID}
    r = requests.get(url = __URL, params = _PARAMS)

```

server.py

```

from wsgiref.simple_server import make_server
from pyramid.config import Configurator
from pyramid.response import Response
from prometheus_client import generate_latest, CONTENT_TYPE_LATEST
from prometheus_client import CollectorRegistry, Gauge
from pyramid.view import view_config
import insert
import database
import selectQ

#This function returns a registry to the prometheus
#Each time a gauge is created the gauge are setted to the metrics
def __initRegistry(T,C,H,V):

```

```

        registry = CollectorRegistry()
        for temps in T:
            gT = Gauge("Temperature", "Temperature Of Solar Panel: ", registry=registry)
            gT.set(temps[1])
            for currents in C:
                gC = Gauge("Current", "Current Of Solar Panel", registry=registry)
                gC.set(currents[1])
                for humidities in H:
                    gH = Gauge("Humidity", "Temperature Of Solar Panel", registry=registry)
                    gH.set(humidities[1])
                    for voltages in V:
                        gV = Gauge("Voltage", "Voltage of the Solar Panel", registry= registry)
                        gV.set(voltages[1])
                return registry
#This path "/expose", when it receives a get requests
#it returns a registry to the prometheus
@view_config(
    route_name = "expose"
)
def expose_metrics(request):
    T,C,H,V = selectQ.select()
    registry = __initRegistry(T,C,H,V)
    return Response(generate_latest(registry),
        content_type=CONTENT_TYPE_LATEST)
#This path "/store", when it receives a get requests
#it stores the metrics to the MySql database
@view_config(
    route_name = 'store'
)
def store_metrics(request):
    if 'metric' in request.params and 'value' in request.params and 'ID' in request.params:
        print request.params['metric']
        print request.params['value']
        print request.params['ID']
        metric = request.params['metric']
        value = int(request.params['value'])
        ID = int(request.params['ID'])
        insert.insert(value,metric,ID)
        return Response(body = "Stored to database",
            content_type=CONTENT_TYPE_LATEST)
    else:
        return Response(body = "Not stored to database")

# For the configuration of the web pyramid server please refer to the
# https://docs.pylonsproject.org/projects/pyramid/en/latest/api/config.html
def config():
    config = Configurator()
    config.add_route('expose','/expose_metrics')
    config.add_route('store','/store_metrics')
    config.scan()
    app = config.make_wsgi_app()
    server = make_server('127.0.0.1', 80, app)

```



```

        return server
#Open the server forever & ever
#Server.sh calls this function
def serve():
    config().serve_forever()

```

selectQ.py

```

#Select Database Query for gathering from everytable most recent timestamp
import database

def select():
    conn = database.createConn()
    cur = conn.cursor()
#Queries
    sqlCurrent = 'SELECT * FROM current INNER JOIN(SELECT id, MAX(time) AS Maxtime FROM
current GROUP BY id) toptime ON current.id = toptime.id AND current.time = toptime.maxtime'
    sqlTemp = 'SELECT * FROM temperature INNER JOIN(SELECT id, MAX(time) AS Maxtime FROM
temperature GROUP BY id) toptime ON temperature.id = toptime.id AND temperature.time =
toptime.maxtime;'
    sqlHumidity = 'SELECT * FROM humidity INNER JOIN(SELECT id, MAX(time) AS Maxtime FROM
humidity GROUP BY id) toptime ON humidity.id = toptime.id AND humidity.time = toptime.maxtime;'
    sqlVoltage = 'SELECT * FROM voltage INNER JOIN(SELECT id, MAX(time) AS Maxtime FROM
voltage GROUP BY id) toptime ON voltage.id = toptime.id AND voltage.time = toptime.maxtime;'
#Execute the select query for the current table
    cur.execute(sqlCurrent)
    current = cur.fetchall()
#Execute the select query for the temperature table
    cur.execute(sqlTemp)
    temp = cur.fetchall()
#Execute the select query for the humidity table
    cur.execute(sqlHumidity)
    humidity = cur.fetchall()
#Execute the select query for the voltage table
    cur.execute(sqlVoltage)
    voltage = cur.fetchall()

    return temp,current,humidity,voltage

```

insert.py

```

#This is the insert python MySql where the values are push to the MySql.
#Called from the python server.
import database
import time
import datetime
def insert(value, metric, id):
    conn = database.createConn()
    cur = conn.cursor()
    ts = time.time()
    timestamp = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')
    if metric == 'C':

```

```

        sql = "INSERT INTO current (ID,value,time) VALUES (%s, %s, %s)"
    elif metric == 'H':
        sql = "INSERT INTO humidity (ID,value,time) VALUES (%s, %s, %s)"
    elif metric == 'T':
        sql = "INSERT INTO temperature (ID,value,time) VALUES (%s, %s, %s)"
    elif metric == 'V':
        sql = "INSERT INTO voltage (ID,value,time) VALUES (%s, %s, %s)"
    value = (id,value,timestamp)
    cur.execute(sql,value)
    conn.commit()
    print(cur.rowcount, "record inserted.")

```

Discovery Service

These commands are for the discovery service between the Xbee's.

The +++ finds the coordinator.

Every AT with ID sends a request to the end device Xbee.

You should get a response acknowledgement of 'ok' for every node.

This is describe in the Digi Documentation:

https://www.digi.com/resources/documentation/Digidocs/90001496/tasks/t_use_at_commands.htm?TocPath=XBee%20transparent%20mode%7CCommand%20mode%7C____2

```
send +++;
```

```
send AT(ID);
```

prometheus.yml (Configuration File)

```
# Local Prometheus for configurations for the SBC
```

```
global:
```

```
    scrape_interval: 5s
```

```
    evaluation_interval: 5s
```

```
# Alertmanager configuration
```

```
alerting:
```

```
    alertmanagers:
```

```
    - static_configs:
```

```
      - targets:
```

```
rule_files:
```

```
scrape_configs:
```

```
    # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
```

```
    - job_name: 'prometheus'
```

```
      # metrics_path defaults to '/metrics'
```

```
      metrics_path: /expose_metrics
```

```
      static_configs:
```

```
      - targets: ['localhost:80']
```

Photographs of Implementation - Proof of work



Image 1: Implementation of the apparatus at Foss Research Center. Data Logger can be seen inside the upper left white box.



Image 2: Dr Makrides cutted a black cable and used it to merge it with the direct current output of the solar panel.

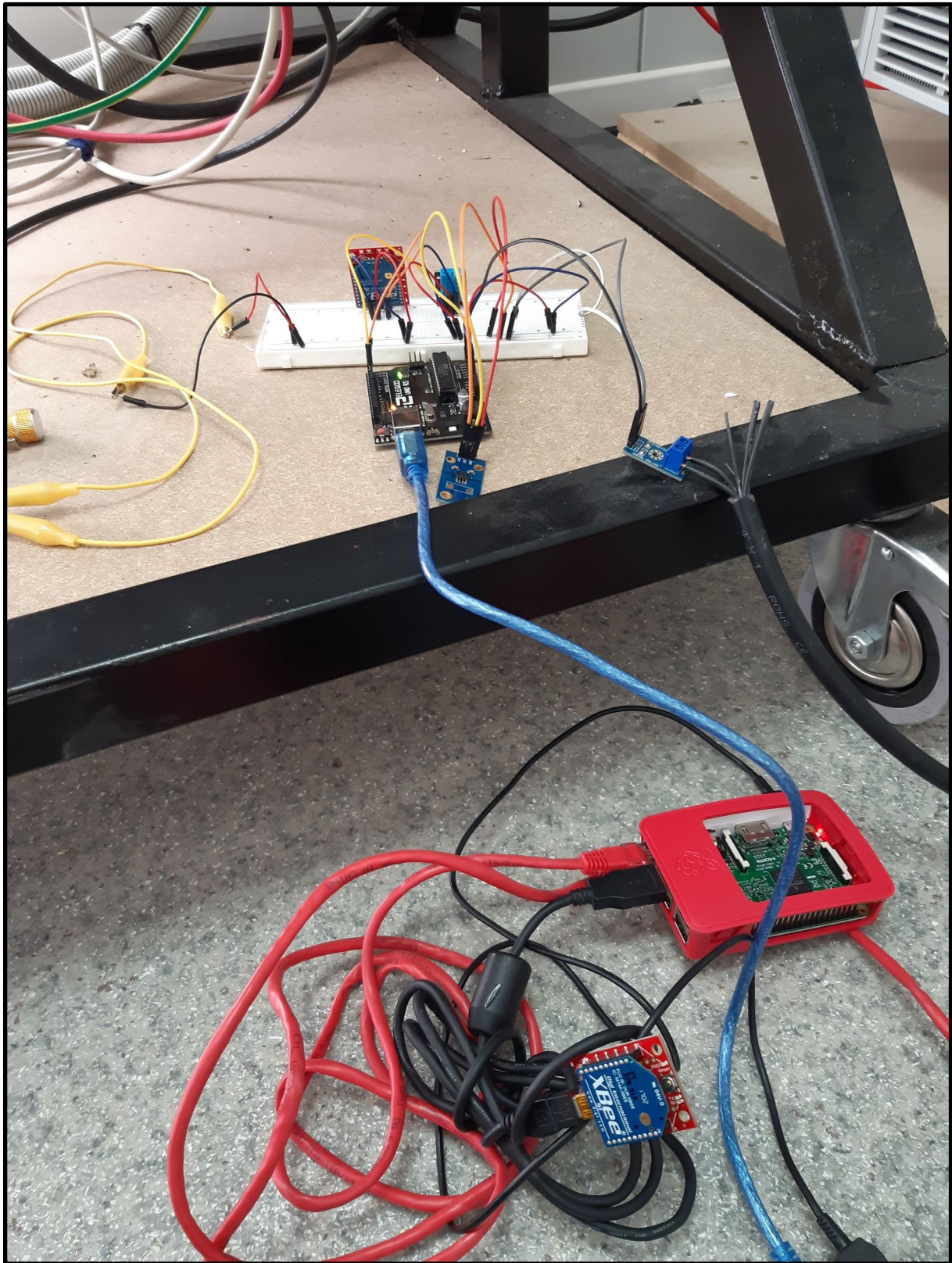


Image 3: The HCMODU0047 sensor is used for sensing the voltage from the black wired that comes directly from the solar panel.

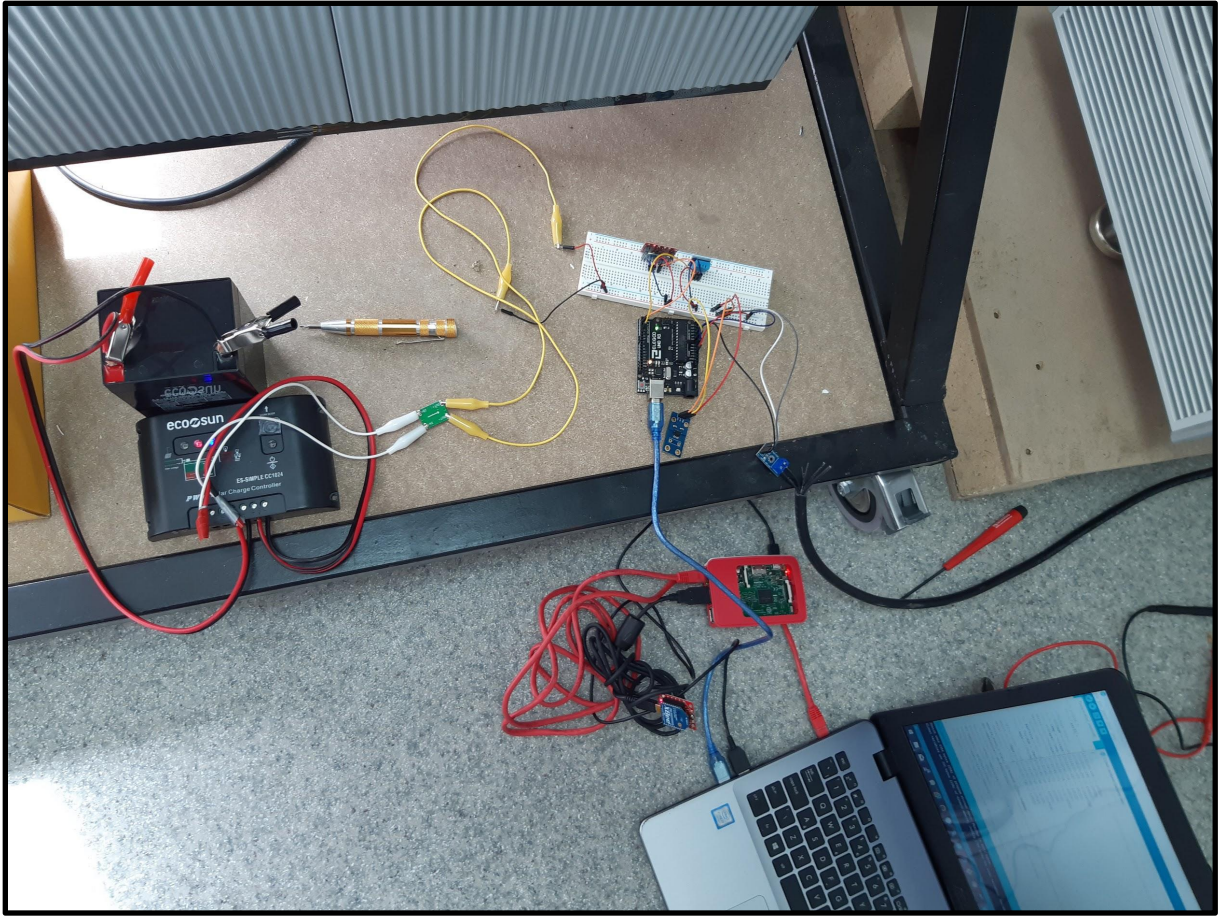


Image 4: Implementation of the Apparatus at Foss Research Center. Power Regulation can be seen as described.

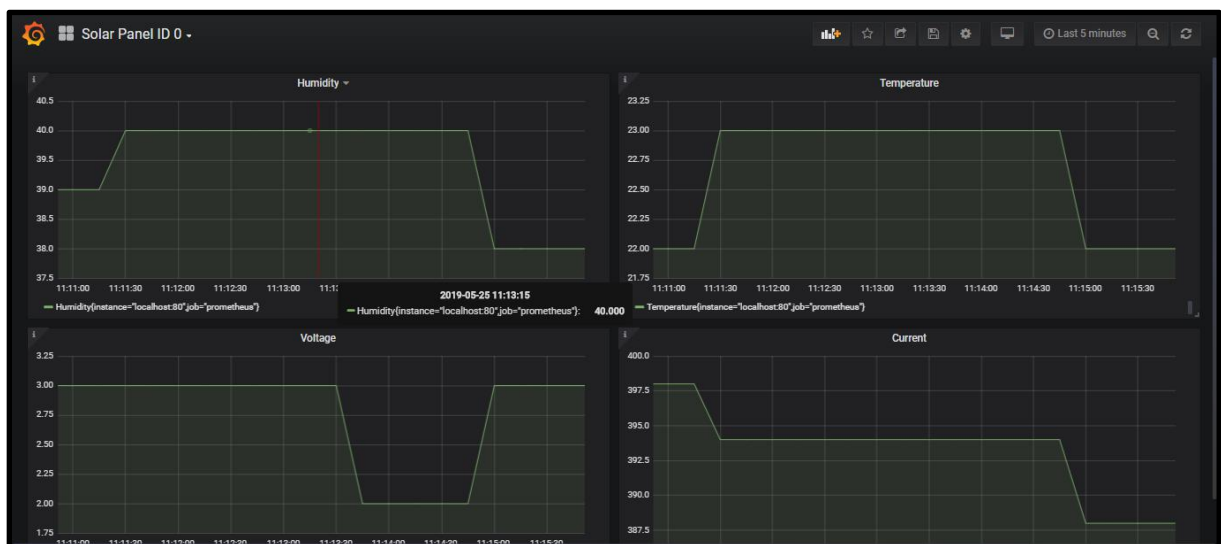


Image 5: Data fetch from the apparatus being visualized using four different timeplot graphs.



Image 6: The solar panel from which the data collecting happened. Dr Makrides indicated that it was the uppermost right.