OPTIMIZATION OF THE INVERTERLESS SOLAR DC SYSTEM

AND

THE MQTT LIBRARY IMPLEMENTATION

A Project Report
Submitted by

K HARIPRIYA
(EE15B090)

In the partial fulfillment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY

&

MASTER OF TECHNOLOGY

in

ELECTRICAL ENGINEERING

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS
CHENNAI-600 036, TAMIL NADU, INDIA

JUNE 2020

THESIS CERTIFICATE

This is to certify that the thesis titled OPTIMISATION OF THE INVERTERLESS SOLAR DC SYSTEM AND THE MQTT LIBRARY IMPLEMENTATION, submitted by K Haripriya, to the Indian Institute of Technology, Madras, for the award of the degree Bachelor of Technology & Master of Technology in Electrical Engineering, is a bonafide record of the research work done by her under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Place: Chennai

Date: 10th July 2020

Prof. Ashok Jhunjhunwala

Project Guide

Professor

Dept. of Electrical Engineering

IIT-Madras, 600 036

ACKNOWLEDGMENTS

Foremost, I would like to express my sincere gratitude to my advisor and project guide Professor Ashok Jhunjhunwala, Electrical Engineering, IIT Madras, for his guidance and continuous support throughout my Dual degree study and project, for his patience, motivation, insightful comments and immense knowledge. Working under him was a pleasant and great learning experience for me and I would like to thank him for giving me this opportunity.

I wish to express my profound gratitude to my co-guides/supervisors Mr S Sree Hari Nagarajan and Mr Durai Bose, Chakra Network Solutions Pvt. Ltd., IITM Research Park, for their valuable guidance throughout my project. Their suggestions, critical comments, and constant encouragement have been instrumental in completing this work.

Last but not the least I would like to thank all the people who extended their support and shared valuable information that helped in the completion of this project.

ABSTRACT

OPTIMIZATION OF THE INVERTERLESS SOLAR DC SYSTEM

KEYWORDS: Inverterless Solar DC system, Power management algorithm, Efficiency, minimized cost

The Inverterless Solar DC system is a decentralized solar rooftop system that is designed to provide DC power at a very low cost. The system combines a Solar PV, a Li-ion battery, and the grid in a highly efficient manner and supplies power to the load. The system is cost-effective and efficient in powering off-grid homes where providing grid connectivity is not economical. In places that have grid connectivity but face load shedding, this system can replace the inverters, to provide uninterrupted power supply at a reduced cost. Even at other places that have grid power available all the time, this system can make the households self-sufficient in sourcing power to meet their energy demand at a lower cost.

The Solar-DC system has three power sources in usual grid connected places which are the Solar PV, the Battery, and the Grid. In places where the grid is not available like in remote areas, the Solar and the Battery support the entire load. Since multiple sources are available at the same time, a decision must be taken as to which source supplies the load or what is the fraction of the load power that is supplied by each of the power sources. Another decision of utmost importance is the charging and discharging of the battery, i.e. at any given point in time, it must be decided whether the battery acts as a source or as a load. These power decisions are handled by the power management algorithm in the firmware code of the system. The effective cost per unit of energy mainly

depends on these decisions, which ultimately decides whether the installation of this system is beneficial or not, especially in grid-connected places. Thus this algorithm must be optimized to make the inverter less system optimal in terms of efficiency, the effective cost per unit, and the emergency power backup hours the battery provides.

This work aimed to come up with an optimal power management algorithm that manages the sources and loads connected to the system efficiently to minimize the effective cost per unit of energy consumed. Two algorithms were proposed and both the proposed algorithms performed better in all terms when compared to the existing algorithm. Simulations showed that on average, algorithm 1 showed a 12% reduction, and algorithm 2 showed a 16% reduction in the effective cost per unit of energy.

THE MQTT LIBRARY IMPLEMENTATION

KEYWORDS: MQTT, Eclipse Paho Embedded C library, wrapper library implementation

MQTT is a communication protocol designed primarily for IoT applications where the memory, power, and bandwidth usage are constrained. It is a lightweight event and message-oriented protocol that enables resource-constrained devices to asynchronously communicate and distribute telemetry information to multiple devices efficiently across constrained networks.

Eclipse Paho Embedded C library is a lightweight, open-source MQTT v3.1.1 implementation for embedded devices. It is a generic implementation that is not reliant on any particular libraries for networking, threading, or memory management. This is done to make the library portable since TCP/IP stacks and multithreading libraries are not standardized. Complete implementation of the MQTT protocol using the Eclipse Paho library in any embedded platform requires the implementation of network calls, timer functions, and memory management functions. Threading functions must be implemented if one wants the MQTT communication to function on a separate thread. Besides, security implementation requires integrating a TLS library and implementing the necessary functions.

This work is an implementation of an abstract, easy to use wrapper MQTT library for STM32 Microcontroller devices using the Eclipse Paho Embedded C library. All the necessary functions were implemented to provide a clean and complete MQTT client library with TLS deployment for optional security.

LIST OF TABLES

LIST OF FIGURES

TABLE OF CONTENTS

OPTIMIZATION OF THE INVERTER LESS SOLAR DC SYSTEM

1. INTRODUCTION

1.1 ABOUT THE INVERTERLESS SYSTEM

The Inverterless Solar DC system is a decentralized solar rooftop system that system combines a Solar PV, a Li-ion battery, and the grid in a highly efficient manner to supply power to the load. The system is cost-effective and efficient in powering off-grid homes in remote parts of the country where the terrain is so foreboding that they are beyond the reach of Electricity Grids. The system has been successfully installed in many remote villages in Manipur, Assam, Meghalaya, and Jammu & Kashmir under the Inverterless solar DC technology project that targets 100 per cent electrification of rural households.

In places that have grid connectivity but face load shedding, this system can replace the inverters, to provide uninterrupted power supply at a reduced cost. Even at other places that have grid power available all the time, this system can make the households self-sufficient in sourcing power to meet their energy demand at a lower cost.

Different variants of the system are available like IL500, IL2500, IL3K, etc. and one can choose the one that best suits their load requirement. The number at the end represents the maximum DC power the system can provide. IL500 is designed to support a solar panel, a battery and grid input, and can provide a maximum of 500 W of power. A

block diagram of the IL500 system is shown in Figure 1.1. IL2500 is a scaled version of IL500 and can support a maximum load of 2500 W. IL3K, as shown in Figure 1.2, is an advanced system and is designed with many additional features like better support for AC loads, exporting power to the grid, adding power sources like generators, etc.

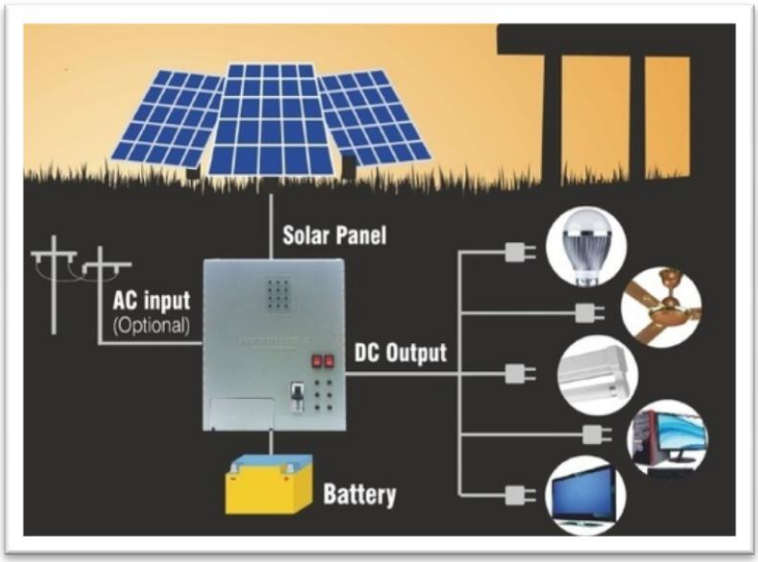


Figure 1.1 The Inverter less 500 Solar-DC System

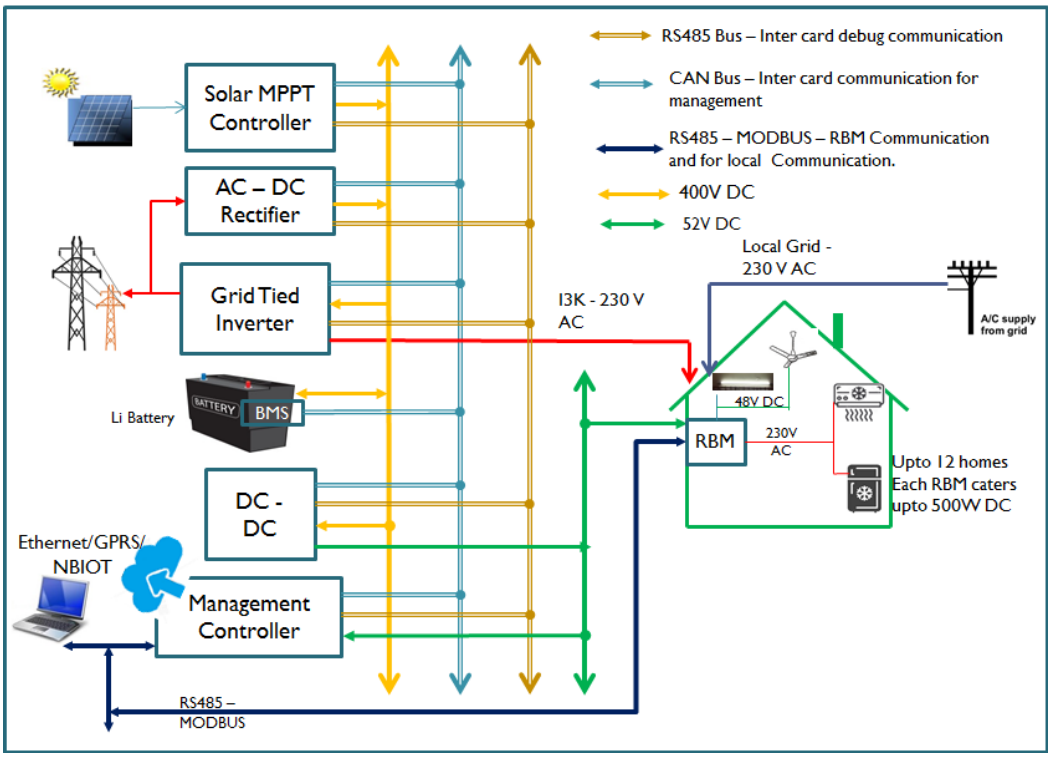


Figure 1.2 IL3K block diagram

1.2 THE INVERTER LESS SOLAR-DC SYSTEM'S POWER MANAGEMENT ALGORITHM

When multiple power sources are available, they have to be managed efficiently to minimize the effective cost per unit, since the cost per unit is different for each power source. Similarly, load management plays an equally important role. Decisions on when to charge the battery and when to discharge it to supply the load have to be taken carefully since the battery life depends on the charge-discharge cycles. Over a period of time (say a day), the power management algorithm of the Inverterless system takes these decisions to ensure that maximum energy is used from solar followed by the battery and is lastly augmented by grid whenever it is available [1]. The algorithm is known as the heartbeat of the system since the efficiency of the system in terms of power loss and in terms of cost per unit of energy, primarily depends on this algorithm. Thus to have an efficient system, the power management algorithm must be optimally written.

2. MOTIVATION AND PROBLEM STATEMENT

In addition to providing low-cost electricity to places where the grid is not set up, the Inverterless system proves to be an efficient and cost-effective alternative to the centralized grid power system. Cost and efficiency are the two key factors that one considers when shifting from the current centralized AC grid system to the decentralized solar PV system. Hence the system has to be optimal in these terms which depend on the power management algorithm as stated earlier.

Problem statement

1) Improvise the power management algorithm in IL500 to optimize the Inverterless system with respect to cost and efficiency.

2) Write an optimized power management algorithm for the IL3K system with the following objectives:

    a. Minimize the cost per unit
        i. Maximize solar energy utilization
            - Maximize direct supply to the load
            - Try to capture the maximum of the excess in battery
            - Export only if the battery is fully charged
        ii. Minimize grid energy imported
            - Maximize direct supply to the load
            - Minimize energy supply from the grid to the battery to the load
        iii. Reduce the losses and improve the overall efficiency

    b. Maintain a certain guaranteed battery energy backup

3. PROPOSED MECHANISM/ALGORITHM

3.1 PREVIOUSLY USED ALGORITHM

The following chart briefly describes the previously used algorithm in the IL500 system. A single battery SoC (State of Charge) threshold is defined (ex. 45%), based on which most of the instantaneous power decisions are taken.

Solar power > load power	**Solar power < load power**
Battery SoC > Threshold • Supply load from Solar and use excess power to charge battery if battery is not full **Battery SoC < Threshold - ΔTh** • Supply load from Solar • Charge battery at a fixed rate until SoC reaches Threshold. Use grid if solar power is insufficient to charge battery.	**Battery SoC > Threshold** • Supply Load from Solar and Battery. • Use battery as source until SoC falls below Threshold – ΔTh **Battery SoC < Threshold - ΔTh** • Supply Load from the Solar and the Grid. • Use grid to charge battery at a fixed rate until SoC reaches Threshold.

The ΔTh is used to introduce hysteresis in the charging/discharging cycle of the battery so that the battery state does not switch back and forth between charging and discharging when the SoC is close to the set threshold value. ΔTh is taken as 5% in the existing algorithm.

3.2 PROPOSED APPROACH

3.2.1 Reduce energy Loss and improve Efficiency

Presently, the IL500 uses the following strategy to maintain a minimum guaranteed power backup. It uses a battery SoC (State of Charge) threshold value (for ex. 50%) based on which the battery charge-discharge decision and other power decisions are taken.

i. If the Battery SoC is lesser than threshold – ΔTh (for ex. 45%), the battery is charged until the SoC goes above the threshold. If solar power is insufficient to supply the load and charge the battery, the remaining requirement is imported from the grid.

ii. If the Battery SoC is above the threshold and the solar power is insufficient to supply the load, the remaining requirement is met by discharging the battery until the SoC goes below threshold - ΔTh.

Because of this, during the non-solar hours, the battery state alternates between (i) and (ii) continuously and the battery SoC oscillates between threshold and threshold - ΔTh. This is shown in Figure 1.3. This leads to unnecessary loss of energy since in (i), the grid charges the battery and in (ii), the battery discharges to supply the load, when the grid can directly supply the load. Moreover, the battery charge/discharge cycles are wasted, which reduces battery life. This in turn increases the effective cost per unit of electricity.

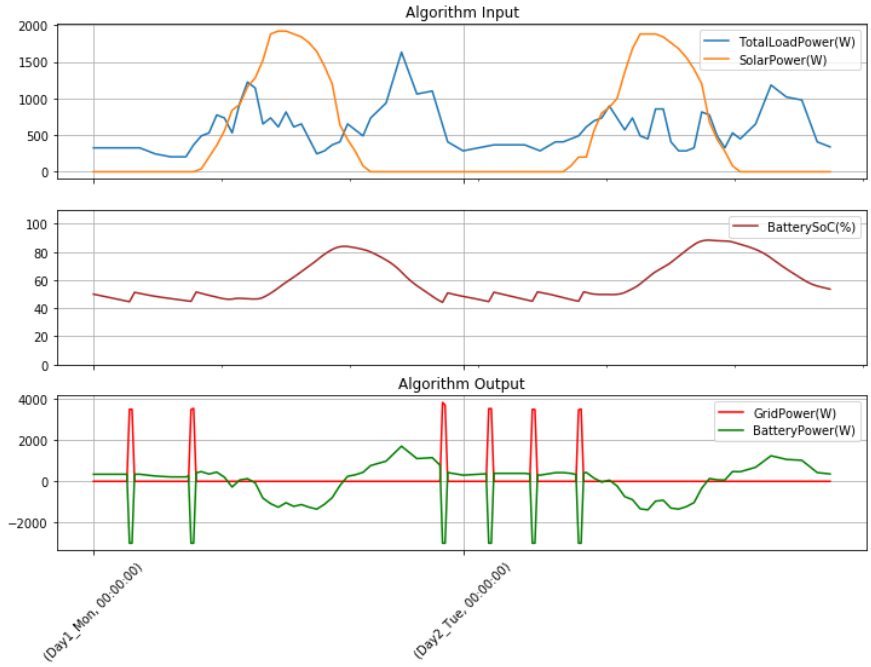


Figure 1.3 Inefficient charge-discharge strategy

Proposed Approach: Modification in the Battery Charge-discharge strategy

i. If the Battery SoC is lesser than threshold - ΔTh, the battery is charged until the SoC goes above the threshold. If solar power is insufficient to supply the load and charge the battery, the remaining requirement is imported from the grid.

ii. If the Battery SoC is above the threshold and the solar power is insufficient to supply the load, the remaining requirement is met by discharging the battery until the SoC goes just below threshold. Thereafter, the Battery SoC is maintained at that level and the remaining load requirement is supplied by the grid.

With this modification, the SoC will go below the threshold - ΔTh only when grid failure or load shedding happens. And when grid power becomes available, the grid will be used to charge the battery until the SoC goes above the threshold. This would reduce the losses and increase the battery life since the grid directly supplies the load instead of cycles of the grid charging the battery and battery supplying the load. Figure 1.4 shows the behavior of the system with the proposed strategy.

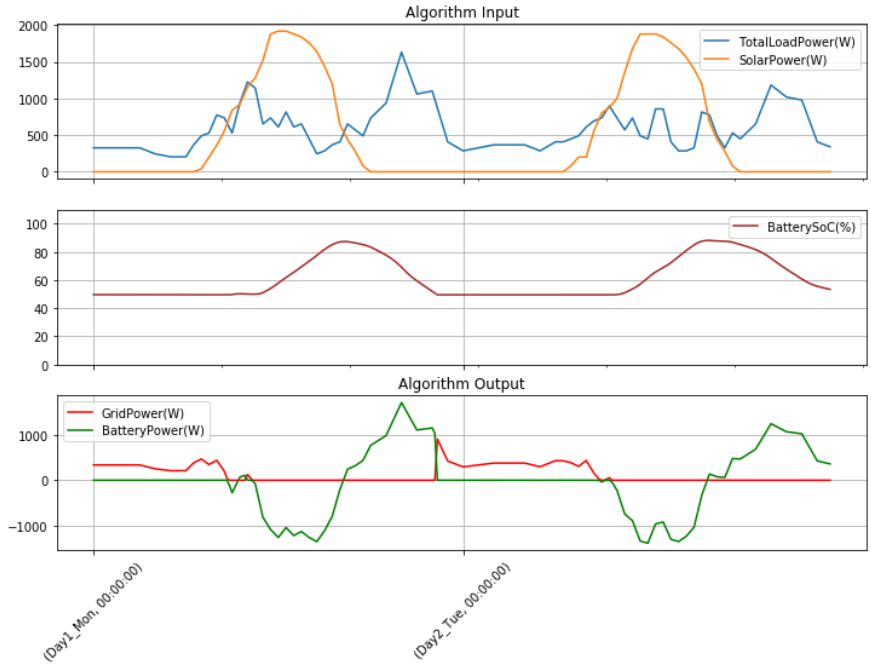


Figure 1.4 Proposed charge-discharge strategy

3.2.2 Trade-off between maximizing solar utilization and maintaining sufficient battery backup

A fixed SoC threshold leads to a trade-off between maximizing solar utilization and maintaining guaranteed battery backup hours. If the Battery SoC threshold is set high (ex. 70%), we get more power backup hours when grid failure or load shedding happens. But during the sun hours when solar provides more power than the load requirement, all the excess solar cannot be stored in the battery since the battery will not be sufficiently empty. This leads to wastage of solar energy or unnecessary energy export to grid (if it is supported) and subsequent energy import from the grid to meet the load requirement. If the Battery SoC threshold is set low (ex. 20%), we can store the available solar energy in the battery and minimize Solar wastage, but at the cost of having lesser power backup hours.

Proposed Approach 1: Have two SoC Thresholds

Instead of having a single SoC threshold, we define two thresholds, an upper threshold (say about 70%) and a lower threshold (say about 40%) for the battery. These are kept configurable so that we can set them based on the load requirement and the solar availability at the place of installment. As we approach the evening (say between 5:30 p.m.-8:30 p.m.), we try to keep the battery close to the upper threshold to ensure more power backup at night time, hence we set the battery SoC threshold value to the defined upper threshold (ex. 70%) in this time range. During the non-sun hours (at night time), we allow the battery to be sufficiently drained so that before the next morning, the SoC is closer to the lower threshold. This would ensure that the wastage of the available solar energy is lesser since we keep the battery sufficiently empty to capture the excess solar energy. Hence we set the battery SoC threshold to the defined lower threshold (ex. 50%) in the other time range (8:30 p.m. to 5:30 p.m.). In any case, we don't allow the battery to be drained below the lower threshold and hence guarantee a certain power back-up hours. In summary, by the end of sun-hours, the SoC will be closer to the upper threshold and before the beginning of the sun-hours, the SoC will be closer to the lower threshold.

This approach guarantees a certain battery energy backup for the non-solar hours but does not solve the problem entirely. This is because one cannot be sure that the battery SoC will reach the lower threshold by morning since it depends on the load requirement during the non-solar hours. The load pattern is not the same for every system and moreover, it changes with changing weather conditions. Hence the thresholds have to be set independently for every system and must be changed manually now and then.

Proposed Approach 2: Auto Update the SoC threshold to the best value

The best threshold is the one that leaves just the minimum required space in the battery to capture the entire excess solar energy available. With this, solar energy would charge the battery to 100% by the end of solar hours giving the maximum battery backup for the non-solar hours. This is illustrated in Figure 1.5.

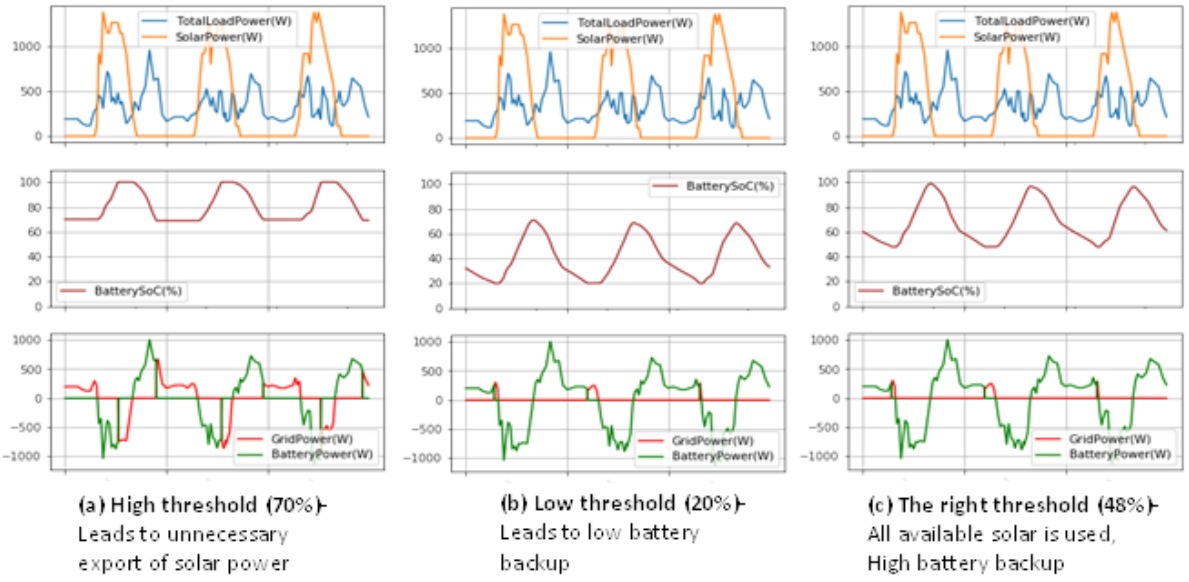


Figure 1.5 Choosing the best threshold value

But this threshold value depends on the solar insolation and the load power profile which varies from day to day and from place to place. So every day, we try to update the threshold to the best possible value. The threshold is set on the previous day, at the right time, so that the battery drains in the night and by morning the SoC reaches the set threshold. This means that the excess solar that is going to be available tomorrow has to be predicted today. Hence at 6:30 p.m. every day, we use the data from the past 12 days to predict the excess solar that is going to be available the next day. A simple AR (Auto-Regressive) model is used for the prediction of the excess solar energy available. The threshold value is calculated using the prediction and the Battery SoC threshold is set to

that value. The performance of the system will therefore depend on how close the prediction is to the actual value.

3.3 Complete Algorithm

Battery SoC threshold is dynamic with time and depends on which of the proposed approaches is used. At time t, let SoC_U = threshold and SoC_L = threshold – ΔTh. At that time,

a. If Battery SoC < SoC_L:

- The battery is charged at a fixed charging rate until the Battery SoC reaches SOC_U (for ex. charging rate = 0.2C ⇒ 2kW for a 10 kWh battery).
- If Excess Solar power (= Solar power – load power) > 2kW, grid is not used since solar gives sufficient power supply the load and charge the battery.
- If Excess Solar power < 2kW, grid is used to charge the battery and grid provides the rest of the power so that battery is charged with 2kW.
- If Excess Solar power < 0, i.e. Solar power < load power, then the solar and the grid are used to supply the load and charge the battery with 2kW.

b. if Battery SoC > SoC_U:

- If Excess Solar power < 0, then the solar and the battery are used to supply the load. The battery is discharged until the SoC drops just below SoC_U. After that the solar and the grid are used to supply the load and the battery is neither charged nor discharged.
- If Excess Solar power > 0, the solar is used to supply the load and charge the battery until the battery is fully charged.

<u>If exporting power to grid is not supported (IL500):</u>

At any time instant, if the battery is not fully charged, the excess solar power is used to charge the battery until the SoC reaches 100%. If excess solar power is more than the

maximum allowed charging power of the battery, the battery is charged with that maximum allowed power.

If exporting power to grid is supported (IL3K):

At any time instant, if the battery is not fully charged, the excess solar power is used to charge the battery until the SoC reaches 100%. If excess solar power is more than the maximum allowed charging power of the battery, the battery is charged with that maximum power and the rest of the excess power is exported to grid. Once the Battery SoC reaches 100%, all the excess solar power is exported to the grid until SoC falls below 98% again, to avoid continuous switching between exporting and charging the battery.

3.4 IMPLEMENTATION DETAILS

3.4.1 IL500 System

The proposed algorithm with two SoC thresholds and the modification in the battery charge-discharge strategy was implemented in the IL500 source code. The code was implemented and built using the IAR EWARM V8 IDE. The firmware was flashed in the IL500 system and the functionality and performance tests were done.

a) **Functionality Tests:** Testing was done at two units, one at Cygni IITM Research Park and another at Cygni Hyderabad Office simultaneously. All the test cases were tested separately by varying the power sources and the load accordingly. The system behavior in each case was verified and validated against the expected behavior.

b) **Performance Tests:** The firmware was flashed in the IL500 unit installed at a house in IITM. The data Energy data and the power data sent to the server from the system were analyzed and studied for performance improvements.

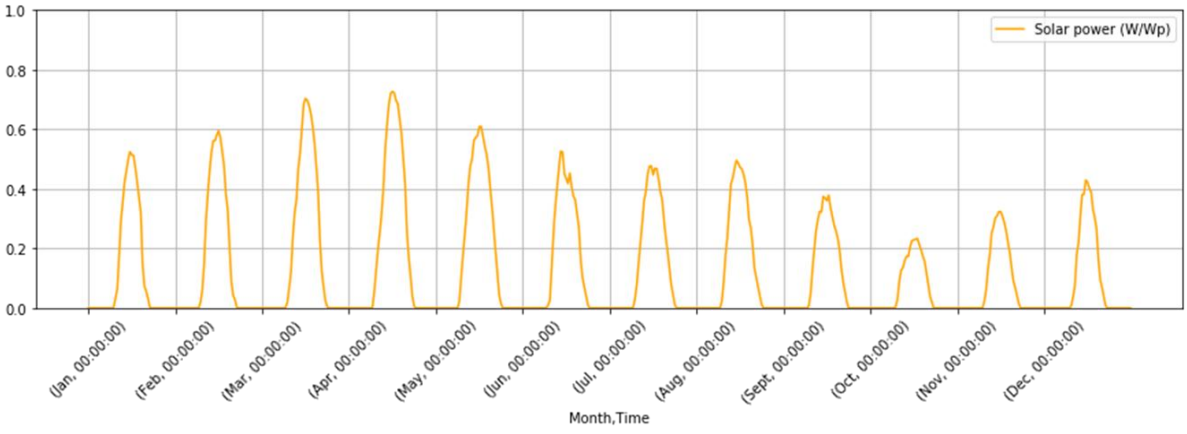
3.4.2 IL3K System

The IL3K system is still in the develop.m.ent phase, hence the system was simulated in Python Jupyter notebook to test and analyze the proposed algorithms. Different solar and load power profiles were constructed for simulation using data collected from the IL500 systems installed at various places. This section presents the data analysis, data processing, and other details on the simulation.

a) Solar Data Analysis:

The solar power data sent by the IL500 unit installed at Professor's home (IITM, Chennai), over the last one year (01 June 2019 to 31 May 2020), was analyzed. The month-wise solar power profile was generated to study the variation of the solar insolation in Chennai. As shown in Figure 1.6**Ошибка! Источник ссылки не найден.Error! Reference source not found.** the maximum insolation was in April, when the peak solar-hours on a day were 4.72 on average. Similarly, the average peak solar-hours was the least, 1.5 in October.

$$peak\ Solar\ hours\ = \frac{Solar\ energy\ generated\ (Wh)}{Solar\ panel\ rating\ (Wp)}$$



Month	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Sun hours	2.87	3.42	4.3	4.72	4.34	3.29	3.24	3.23	2.48	1.5	1.9	2.25

Figure 1.6 Month wise solar power profile in Chennai

b) Solar Power Profiles for Simulation:

Three different solar power profiles, under different weather conditions, namely sunny, cloudy, and rainy were considered for simulations. Figure 1.7 shows the power data plot for each of the profiles, for 4 weeks. Over the 4 weeks, the average peak solar hours in a day for the sunny, cloudy and rainy solar profiles are 4.82 Wh/Wp, 3.24 Wh/Wp and 1.57 Wh/Wp respectively as shown in

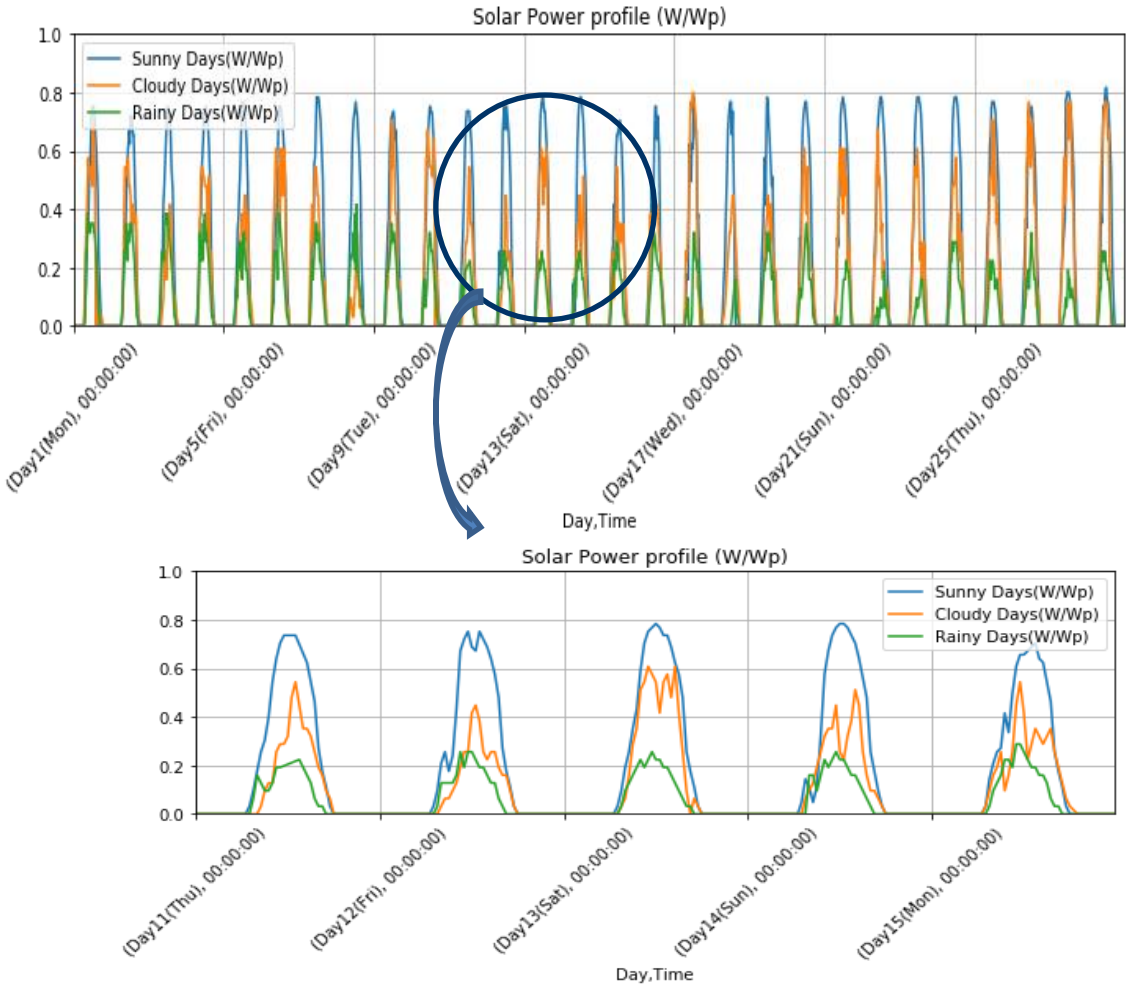


Figure 1.7 Solar power profiles

c) Load power profiles:

Three different load profiles, one household load, and two office loads were considered for simulations. The two office loads differ with respect to the number of working days in a week and the percentage of the load energy consumed during the solar hours.

i. **Household Load:** Data was collected for 4 Weeks from Professor's home IL500 unit and was scaled for IL3K. On average, 44% of the load is during the solar hours, i.e. when the sun is up, as shown in Table 1.1.

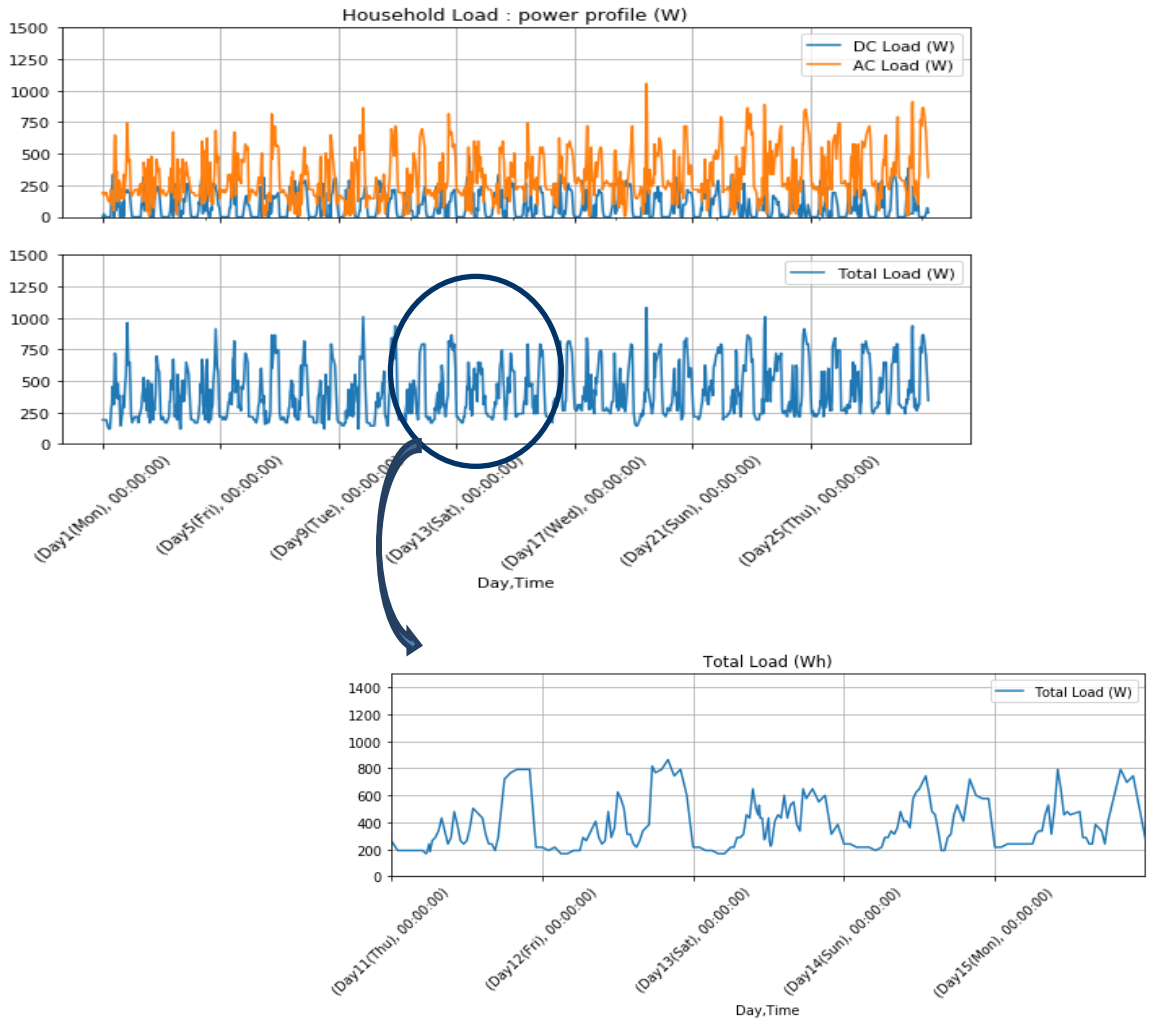


Figure 1.8 Household load power profile

Table 1.1 Household Load energy

	Total Energy (Wh)	Load During Solar hours (Wh)	% Load during Solar hours
Day1(Mon)	8660	4253.33	0.49
Day2(Tue)	7966.67	3926	0.49
Day3(Wed)	8341.33	4053.33	0.49
...	...	...	...
Day26(Fri)	11384	4510	0.4
Day27(Sat)	11386	4607.33	0.4
Day28(Sun)	11251.33	4320	0.38
Mean	**9807.55**	**4265.7**	**0.44**

ii. **Office load 1:** Data was collected for 4 Weeks from a unit at Ashok Ranka and co and scaled for IL3K. The total load was split as 60:40 for AC and DC loads.

Figure 1.9 Office 1 load power profile

This profile has lesser energy consumption on Sundays when compared to the other days in a week. On average, 63% of the load is during the solar hours as shown in Table 1.2.

Table 1.2 Office 1 Load energy

	Total Energy (Wh)	Load During Solar hours (Wh)	% Load during Solar hours
Day1(Mon)	23128	13605.33	0.59
Day2(Tue)	25418	15046	0.59
Day3(Wed)	22306	15238.67	0.68
...	...	...	...
Day26(Fri)	19002	13090.67	0.69
Day27(Sat)	14228	11546	0.81
Day28(Sun)	2452	628	0.26
Mean	**19748.36**	**12683.33**	**0.63**

iii. **Office load 2:** Data was collected for 4 Weeks from a unit at ECIL Kerala and scaled for IL3K. The total load was split as 60:40 for AC and DC loads. The power profile is similar on all the days of the week, as shown in the plot. As shown in Table 1.3, only 26% of the load is during the solar hours on average while the majority of the load is during the non-solar hours.

Table 1.3 Office 2 Load energy

	Total Energy (Wh)	Load During Solar hours (Wh)	% Load during Solar hours
Day1(Mon)	17961.05	4469.33	0.25
Day2(Tue)	14066.95	4472	0.32
Day3(Wed)	12044	4216	0.35
...	...	...	...
Day26(Fri)	11236	2170.67	0.19
Day27(Sat)	15652	4304	0.27
Day28(Sun)	17768	4570.67	0.26
Mean	**13503.71**	**3597.71**	**0.26**

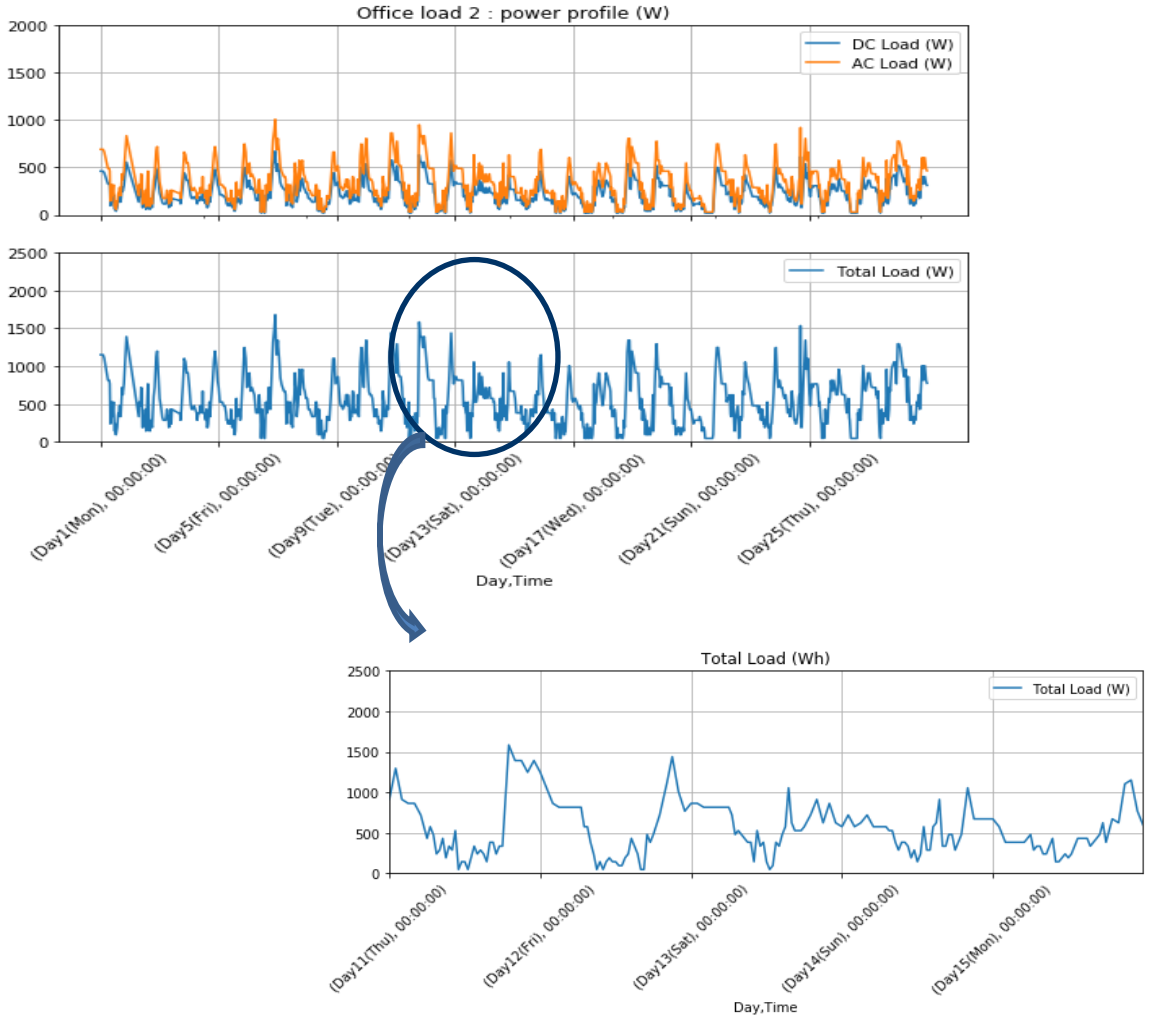


Figure 1.10 Office 2 load power profile

d) System sizing

Sizing the solar panel and the battery correctly is very important for the system to be beneficial. The solar panel must be sized such that the solar energy generated is sufficient to drive the entire day's load on an average day when it is neither too sunny nor rainy. Similarly, the battery must be sized large enough to store the excess solar energy generated during the solar hours, so that it can be used during the non-solar hours. Additionally, one can include the backup energy required when sizing the battery.

i. **Solar Panel Sizing:** Entire load energy must be supplied by solar on an average day.

$$Total\ Load\ Energy\ \ (kWh) = \ peak\ solar\ hours * \ solar\ panel\ rating\ (kWp)$$

ii. Battery Sizing: The battery must store the energy required during non-solar hours and the required emergency backup energy (during power cuts).

$$Battery\ capacity\ (kWh) = \ Total\ Load\ Energy(kWh) * (\ 1 - \%Load\ during\ Solar\ hours) + backup\ required$$

The sizing for the three load profiles, considered for simulation is shown in Table 1.4. The average solar peak hours is taken as 3.24, which is the average calculated during cloudy days. To account for losses, 0.1 is added to the battery size calculation.

Table 1.4 Solar panel and battery sizing calculations

	Solar and battery sizing							
	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**
	Avg daily load (kWh)	**% Load during Solar hours**	**Emergency Battery backup (% of a day's load)**	**Solar Panel rating (kWp) calculated**	**Solar Panel rating (kWp) chosen**	**Battery Sizing (kWh) calculated**	**No of 5 kWh batteries**	**Battery pack capacity (kWh)**
				A / Average solar peak hours		**A x (1 - B + C + 0.1)**		
Household Load	9.81	44.00%	30.00%	3.03	3.00	9.42	2.00	10.00
Office Load 1	19.75	63.00%	20.00%	6.10	6.00	13.23	3.00	15.00
Office Load 2	13.50	26.00%	20.00%	4.17	4.00	14.04	3.00	15.00

e) Losses assumed for Simulation

Energy loss is taken to be 5% if AC to DC or DC to AC conversion is required. Otherwise, the loss is taken to be 2.5%. This is shown in Table 1.5. Similarly, it is assumed that the battery undergoes self-discharge at a rate of 5% per month.

Table 1.5 Losses assumed for simulation

From	To	% loss assumed
Solar	DC loads	2.5%
	AC loads	5%
	Battery	2.5%
	Grid export	5%
Grid	DC loads	5%
	AC loads	2.5%
	Battery	5%
Battery	DC loads	2.5%
	AC loads	5%

4. PERFORMANCE STUDY

4.1 PERFORMANCE METRICS

The following metrics were used to study and compare the performance of the system.

4.1.1 Power Utilization factor for each of the Sources

The power utilization factor for a particular power source measures the percentage of the total input energy supplied by that source. We calculate it on a daily basis. The objective is to maximize solar utilization and minimize grid utilization.

$$Total\ input\ energy = Solar\ Energy - Grid\ export + Grid\ Import$$

$$Solar\ Utilisation\ = (Solar\ Energy - Grid\ export)/Total\ input\ energy$$

$$Grid\ Utilisation\ =\ Grid\ energy\ Import/\ Total\ input\ Energy$$

4.1.2 Power loss and Efficiency

The power loss measures the difference in the supplied energy and the consumed energy. The efficiency calculated is with respect to the power loss.

$$Battery\ as\ Source\ (Wh) = max\left(0, \left(\frac{BattSoC(StartofDay) - BattSoC(EndofDay)}{100}\right) * BatteryCapacity\right)$$

$$Battery\ as\ Load\ (Wh)\ =\ max\left(0, \left(\frac{BattSoC\ (EndofDay) - BattSoC\ (StartofDay)}{100}\right) * BatteryCapacity\right)$$

$$Total\ Source\ Energy\ (Wh)\ =\ Solar\ Energy + Grid\ Import\ Energy + Battery\ as\ Source$$

$$Total\ Used\ Energy\ (Wh) =\ Total\ Load\ Energy + Grid\ Export\ Energy + Battery\ as\ Load$$

$$Loss\ (Wh) =\ Total\ Source\ Energy\ -\ Total\ Used\ Energy$$

$$Efficiency\ =\ (Total\ Used\ Energy\ /\ Total\ Source\ Energy) * 100$$

4.1.3 Effective Cost per unit

The effective cost per unit of energy is the major factor one considers when installing this system at their home or workplace. This is what decides how profitable the system is, especially in grid-connected places. The power utilization factors of the sources, power loss, and efficiency have a direct effect on the cost per unit.

$$
\begin{aligned}
&Total\ Cost \\
&\quad = (Total\ Solar\ Energy\ *\ Solar\ Cost\ per\ Unit) \\
&\quad + (Total\ Grid\ Import\ Energy\ *\ Grid\ in\ Cost\ per\ unit) \\
&\quad + (Total\ Battery\ out\ Energy * Battery\ cost\ per\ unit) \\
&\quad - (Total\ Grid\ Export\ Energy\ *\ Grid\ out\ Cost\ per\ unit)
\end{aligned}
$$

$$Effective\ Cost\ per\ Unit\ =\ Total\ Cost\ /\ Total\ Load\ Energy$$

4.1.4 Average battery backup available at the end of solar hours

The battery SoC at the end of the solar hours (for ex. 6:00 p.m.) is recorded every day to measure the energy backup available for the rest of the day. This factor is of utmost importance in places where the grid is not available or in places that face load shedding.

4.2 RESULTS AND ANALYSIS

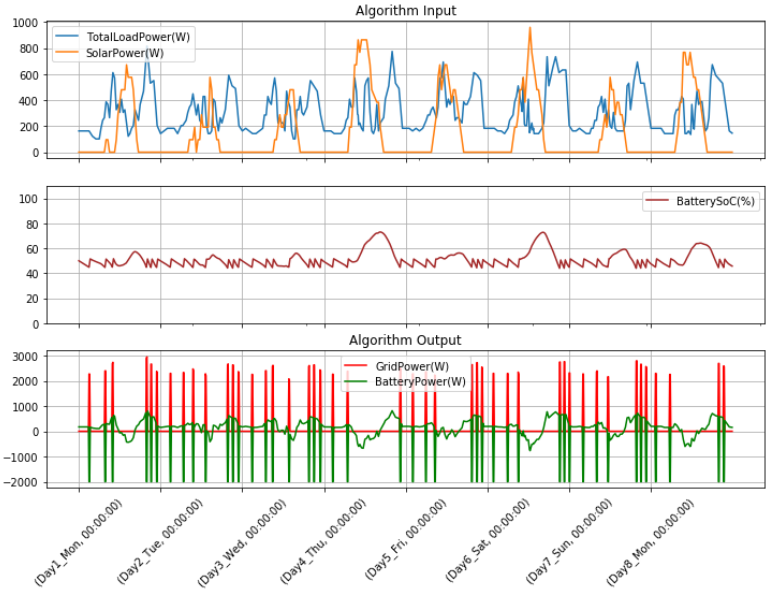
4.2.1 IL500 System Results

The IL500 system flashed with the updated algorithm had many other changes done around the same time. The solar panel capacity was doubled from 125Wp to 250Wp and AC loads were connected to the system which increased the load by a huge amount. Hence the changes in the performance of the system due to the proposed algorithm could not be analyzed. Moreover, the algorithm inputs like solar energy and load energy do not remain constant and change from day to day, thus the results from a real setup do not precisely reflect the performance changes caused by the algorithm.

4.2.2 IL3K System Simulation Results

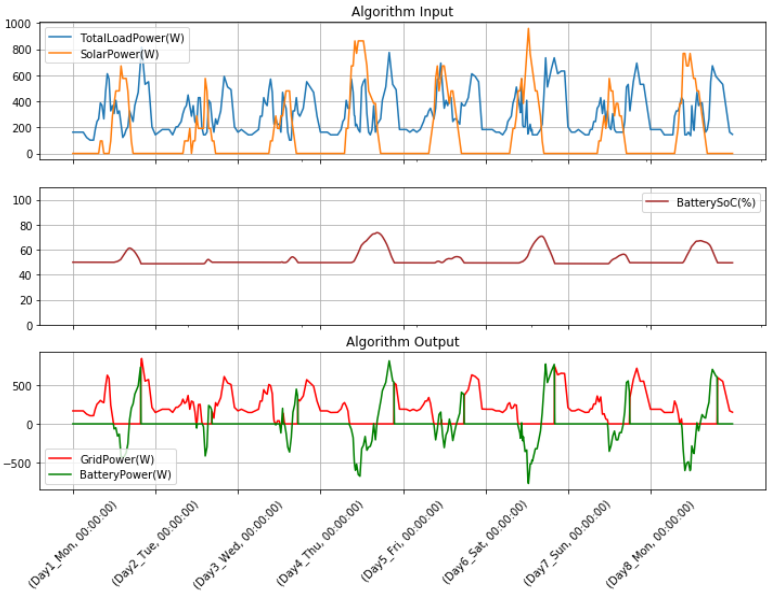
a. Reduction in power loss: Modification in the charge-discharge strategy

The IL3K system with the household load profile and rainy solar profile was simulated for 8 days, with and without the proposed changes in the battery charge-discharge strategy. Rainy solar profile was chosen so that the reduction in power loss can be observed since the grid usage is highest during rainy days when solar generation is less. Figure 1.11 and Figure 1.12 show the power data plot and the power loss and efficiency for the two algorithms. As expected, a huge difference is observed after the proposed changes were made, with a 42.6% reduction in power loss during rainy days. Similarly, simulations done with office load 1 and office load 2 showed 35.2% and 41.8% reduction in power loss respectively.



	Loss(Wh)	Efficiency
Day1_Mon	583.39	92.66
Day2_Tue	566.55	92.26
Day3_Wed	579.58	92.49
Day4_Thu	533.12	93.31
Day5_Fri	613.24	92.99
Day6_Sat	610.89	93.07
Day7_Sun	581.98	92.73
Day8_Mon	527.79	93.14
Mean [Day2:Day8]	**573.31**	**92.86**

Figure 1.11 Simulation with the old charge-discharge strategy



	Loss(Wh)	Efficiency
Day1_Mon	316.47	95.88
Day2_Tue	266.39	96.27
Day3_Wed	273.41	96.31
Day4_Thu	400.26	94.89
Day5_Fri	345.15	95.93
Day6_Sat	369.42	95.69
Day7_Sun	300.96	96
Day8_Mon	345.73	95.4
Mean [Day2:Day8]	**328.76**	**95.78**

Figure 1.12 Simulation using the proposed strategy

b. Manage trade-off between solar utilization and battery backup: With two SoC thresholds

The IL3K system with different load and solar profiles was simulated for 28 days (4 Weeks) in each case. The simulations were done with the previously used algorithm with a single fixed threshold and the proposed algorithm with two SoC thresholds, the upper threshold as 70%, from 5:30 p.m. to 8:30 p.m. and lower threshold as 40% the rest of the day.

The results obtained for the household load are given in Table 1.6. The proposed algorithm with two SoC thresholds performs better in most of the cases, in terms of solar utilization. Additionally, the algorithm guarantees a battery backup of at least 70% at 8:30 p.m. every day, if the grid power is available from 5:30 p.m. to 8:30 p.m.. But since the grid is used to charge the battery to 70% in the evenings, losses are higher and efficiency is reduced.

Similarly, the proposed algorithm showed better performance in terms of the solar utilization and battery backup for the office loads also, but with a slightly reduced efficiency.

Table 1.6 Performance results of the algorithm with two SoC thresholds

Performance results for household load					
Solar Profile	**Algorithm**	**Solar Utilization (%)**	**Grid Utilization (%)**	**Battery Backup At 8:30 p.m. (%)**	**Efficiency (%)**
Sunny	**Single Threshold (50%)**	90.20	9.80	76.88	94.02
	Single Threshold (40%)	96.51	3.49	76.89	93.72
	Two Thresholds	96.51	3.49	76.89	93.72
Cloudy	**Single Threshold (50%)**	80.03	19.97	69.84	94.58
	Single Threshold (40%)	85.08	14.92	65.31	94.46
	Two Thresholds	84.45	15.55	72.38	94.21
Rainy	**Single Threshold (50%)**	46.33	53.67	51.30	95.88
	Single Threshold (40%)	46.33	53.67	41.26	95.88
	Two Thresholds	45.83	54.17	69.21	94.59

c. Manage trade-off between solar utilization and battery backup: With Auto updating SoC threshold

The IL3K system with the different load profiles was simulated for 112 days (16 Weeks) in each case. The simulations were done with the previously used algorithm with a single fixed threshold and the proposed algorithm with an auto-updating SoC threshold, and the results were compared.

A variation was introduced in the solar profile to observe the changing threshold value with changing solar insolation. The variation is in the following manner: Sunny-Cloudy-rainy-Cloudy-Sunny as shown in Figure 1.13. Note that this does not accurately represent a real setup since the solar data constructed for simulation has almost 50% of rainy days which is not the case in reality. The construction is done in this manner only to observe the learning during different weather conditions and the transitions in between.

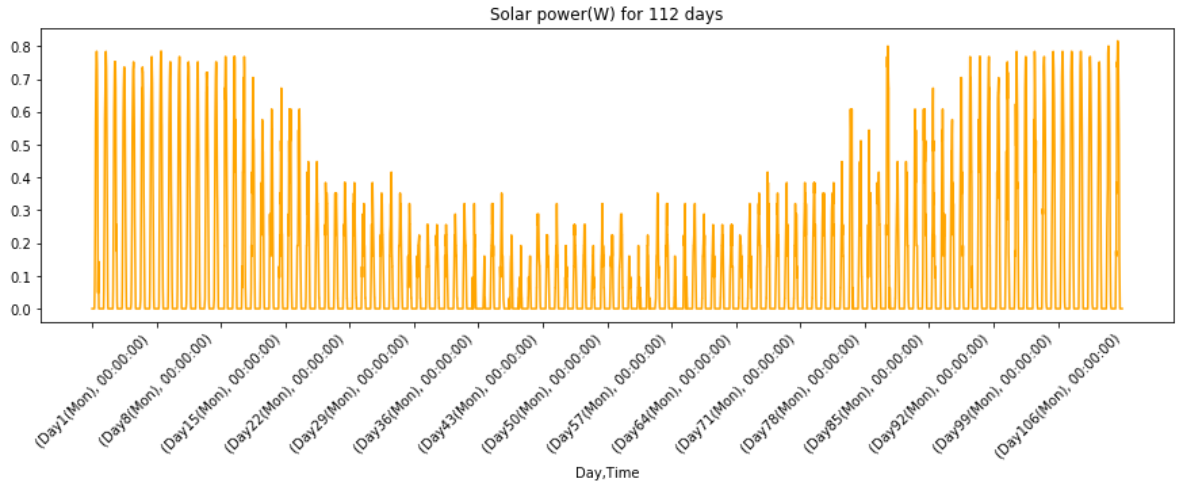


Figure 1.13 Solar variation

i. Prediction results:

Two different AR prediction models were used based on whether there is a weekly pattern observed in the load profile or not. Prediction model A was used for household load and office load 2 which do not show any weekly pattern, and prediction Model B was used for office load 1 which shows a weekly pattern with a lesser load on Sundays. The python code for the two prediction models is given below.

Prediction Model A:

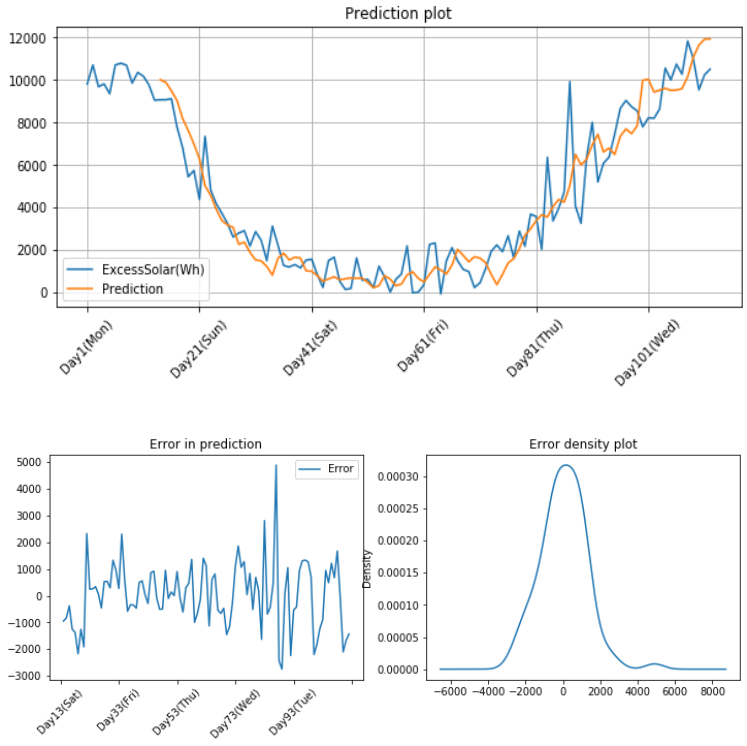
```
trend = (PastData[-5:] - PastData[-12:-7])
Prediction = PastData[-10:-3].mean() + trend.mean()
```

Prediction Model B:

```
trend = (PastData[-5:] - PastData[-12:-7])
Prediction = PastData[-7] + trend.mean()
```

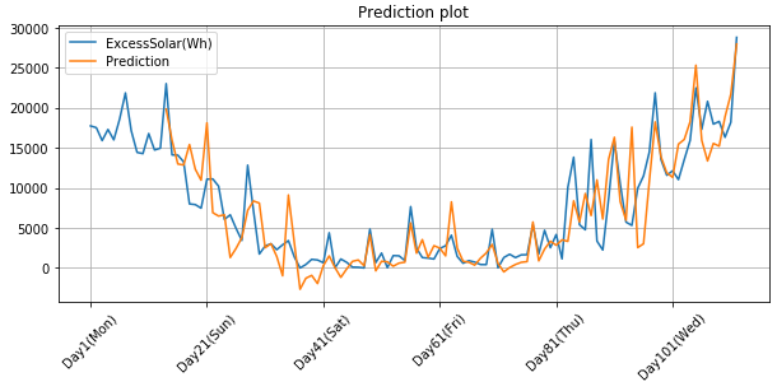
A plot of the predicted values obtained using the prediction model A and actual values of the excess solar energy available, for household load, is shown in Figure 1.14. The error plot and the kernel density plot for the household load show that the error is centered close to zero with a mean error value of 34.23 Wh. The standard deviation of the error is 1220.92 Wh which is 12.4% of the average load in a day. This shows that the predictions are very close to the actual values. The simulation for office load 2 with prediction model A also gave similar results.

Similarly, the prediction results for office load 1 with prediction model B, is shown in Figure 1.15. The peaks in the excess solar on Sundays can be observed due to the lesser load requirement as compared to the other weekdays. The model also predicts higher values on Sundays as shown in the prediction plot.



	Error
count	99.00
mean	34.23
std	1220.92
min	-2747.23
25%	-625.83
50%	46.40
75%	853.66
max	4899.57

Figure 1.14 Household load: Prediction and error in prediction



	Error
count	99.00
mean	202.96
std	3456.90
min	-12276.23
25%	-1133.53
50%	359.05
75%	1951.01
max	9500.06

Figure 1.15 Office load 1: Prediction and error in prediction

ii. Performance results:

The power data plots in Figure 1.16 and Figure 1.17 of household load show that with the proposed algorithm, the battery SoC threshold automatically updates itself based on

the available solar energy and the load requirement to ensure that we maintain high battery backup while maximizing solar utilization at the same time.

The Battery SoC threshold (shown as SoC_L2 in the plot) is lower on sunny days, allowing most of the solar energy to be captured in the battery and to be used in the non-solar hours of the day. Power cut does not make any difference on sunny days since solar supplies the entire load requirement of the day, hence low threshold will not be a problem. The Battery SoC threshold is higher on rainy days since solar energy availability is consistently less. Battery need not have too much room to capture the available solar energy. Moreover, the probability of an emergency power cut is higher on rainy days, which means, higher the battery backup is, the better it is.

The performance results obtained for the household load are shown in Table 1.7. The proposed algorithm performs much better than the old algorithm with a single fixed threshold value, as shown. In the old algorithm, reducing the threshold value improves the solar utilization, but reduces the battery backup available. Whereas, in the proposed algorithm, there is an improvement in both average solar utilization and average battery backup available at the end of solar hours. Similar results were obtained for the office loads as well.

Table 1.7 Performance results of the algorithm with auto-updating threshold

Results for Household load			
	Solar utilization (%)	**Grid utilization (%)**	**Battery backup @ 6 p.m. (%)**
Old algorithm – single fixed threshold (55%)	64.45	35.55	77.23
Old algorithm – single fixed threshold (45%)	68.03	31.97	71.27
proposed algorithm – Auto updating threshold	70.04	29.96	86.13

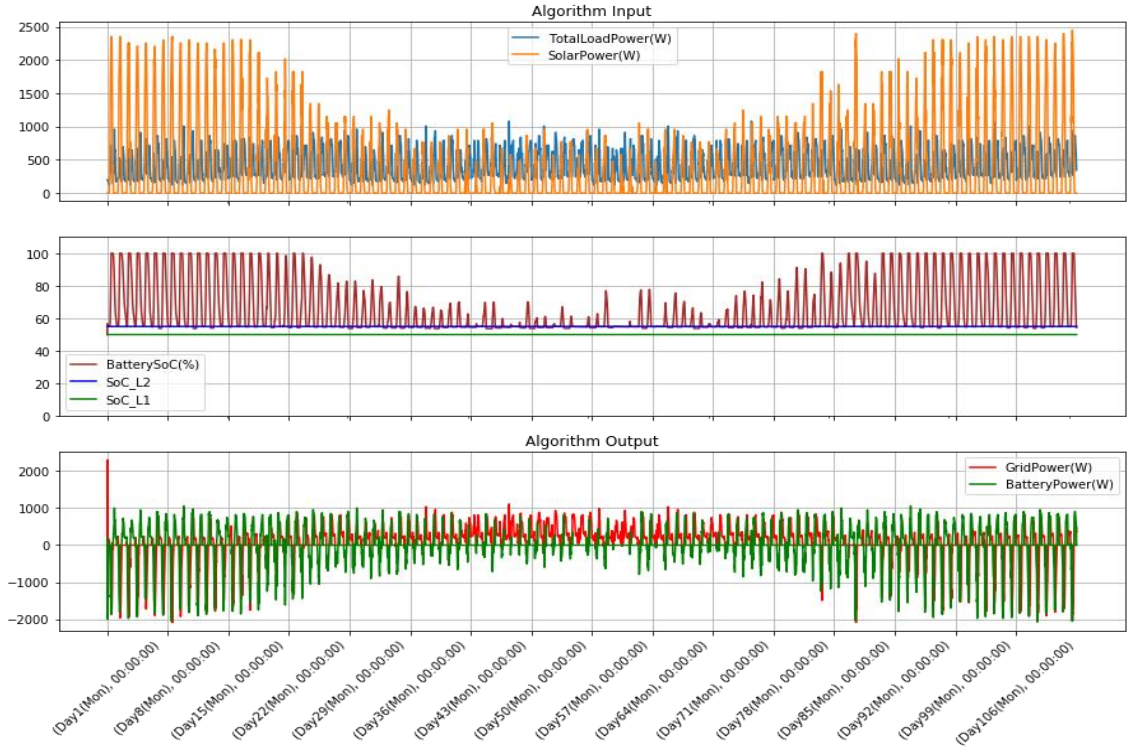


Figure 1.16 Simulation with fixed SoC threshold (55%)

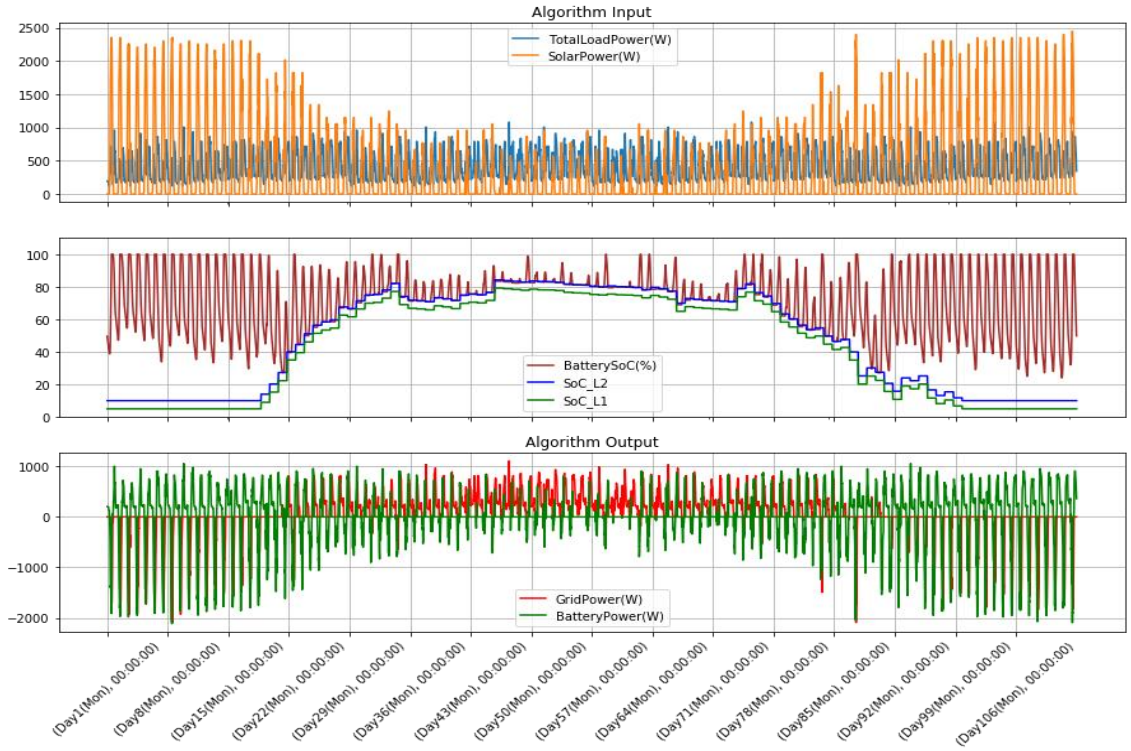


Figure 1.17 Simulation with auto-updating threshold

d. Overall Results

The numbers in the previous sections do not reflect the reality precisely since they were done only for specific solar profiles. Hence, the simulations were done for an entire year to get results close to reality. The assumed cost per unit of energy for each source is as follows:

a) Solar Cost per Unit = ₹ 2.5
b) Grid In Cost per Unit For Household loads = ₹ 6.0
c) Grid In Cost per Unit For Office loads = ₹ 10.0
d) Grid Out Cost per Unit = ₹ 3.5

Table 1.8 summarises the results obtained using the old and the proposed algorithms for the three load profiles considered. It is evident from the table that in all the three load profiles, algorithm 2 with the auto-updating threshold gives the best performance results in terms of both cost and efficiency. In terms of the average battery backup available at the end of solar hours (measured at 6:00 p.m.), although all the three algorithms give similar results, algorithm 1 guarantees a minimum of 70% (or the set upper threshold) at 8:30 p.m. in case of uninterrupted grid supply between 5:30 p.m. to 8:30 p.m., whereas the algorithm 2 does not. Algorithm 2 will maximize the backup available at the end of solar hours on a particular day, only if the prediction was close to the actual excess solar energy on that day.

Table 1.8 Complete simulation results

	Algorithm Used***	% solar utilization	Battery backup @ 6 p.m. (%)	Loss (Wh)	Efficiency (%)	Effective cost per unit cost Assuming the battery cost per unit is			
						₹ 2.00	₹ 4.00	₹ 6.00	₹ 8.00
Household load	0	75.11	82.49	821.31	93.33	₹ 4.80	₹ 6.07	₹ 7.35	₹ 8.62
	1	78.99	81.18	700.48	94.14	₹ 4.39	₹ 5.41	₹ 6.43	₹ 7.46
	2	79.23	82.33	655.92	94.58	₹ 4.24	₹ 5.15	₹ 6.06	₹ 6.97
office load 1	0	74.60	70.43	1465.50	94.26	₹ 5.67	₹ 6.62	₹ 7.57	₹ 8.53
	1	75.83	72.42	1300.60	94.89	₹ 5.29	₹ 5.98	₹ 6.68	₹ 7.37
	2	77.69	71.39	1221.62	95.22	₹ 5.08	₹ 5.72	₹ 6.35	₹ 6.99
office load 2	0	69.00	88.54	1187.74	93.34	₹ 6.70	₹ 8.22	₹ 9.73	₹ 11.24
	1	74.55	86.62	966.74	94.33	₹ 5.85	₹ 6.98	₹ 8.12	₹ 9.25
	2	77.66	86.02	916.99	94.50	₹ 5.58	₹ 6.73	₹ 7.89	₹ 9.04

***** 0:** Old algorithm with a single SoC threshold (50%) and old charge-discharge strategy
1: Algorithm with two SoC thresholds (70% and 40%) and the proposed charge-discharge strategy
2: Algorithm with auto-updating SoC threshold and the proposed charge-discharge strategy

4.3 SUMMARY OF PERFORMANCE STUDY

As shown in Table 1.9, both the proposed algorithms perform better in all terms when compared to the existing algorithm but the algorithm with the auto-updating threshold gives the best performance results. When compared to the existing algorithm with a 50% SoC threshold, on average, algorithm 1 showed a 3.55% increase in solar utilization, while algorithm 2 showed a 5.29% increase. Similarly, algorithm 1 showed a 0.81% increase in efficiency, while algorithm 2 showed a 1.12% increase, on average. In terms of the effective cost per unit of energy, on average, algorithm 1 showed a 12% reduction, while algorithm 2 showed a 16% reduction.

Table 1.9 Performance Summary

Algorithm***	Increase in % solar utilization	Increase in % efficiency	% reduction in Cost per unit
1	3.55	0.81	12%
2	5.29	1.12	16%

***** 1:** Algorithm with two SoC thresholds (70% and 40%) and the proposed charge-discharge strategy
2: Algorithm with auto-updating SoC threshold and the proposed charge-discharge strategy

5. CONCLUSIONS AND FUTURE WORK

The work started with the motive to come up with an optimal power management algorithm that manages the sources and loads connected to the system efficiently to minimize the effective cost per unit of energy. Solar utilization factor and efficiency of the system are the two key factors that decide the cost per unit. The solar utilization can be maximized by keeping the battery empty before the solar hours but this leaves zero battery power backup for emergency power cuts or load shedding. Hence the work primarily aimed to improve the solar utilization factor while maintaining sufficient battery backup and to decrease the losses and improve the overall efficiency of the system.

The proposed algorithm with the auto updating threshold along with the proposed modification in the battery charge-discharge strategy resulted in a 16% reduction in the cost per unit of energy, on average. The efficiency of the system also went up by a noticeable amount. As stated earlier, with this algorithm, the performance of the system highly depends on how close the prediction is to the actual value of the excess solar energy available. Hence with better prediction models, the performance may improve further, resulting in a further reduction in the cost. Moreover, the prediction models proposed here have fixed AR coefficients and hence the performance may vary from system to system. To get the best prediction model for a particular system, ARIMA (Auto-Regressive Integrated Moving Average) models can be trained using the data collected from that specific system. One can even have separate prediction models for solar energy and the load energy during solar hours and combine the results to get the prediction for the excess solar energy.

THE MQTT LIBRARY IMPLEMENTATION

1. INTRODUCTION

1.1 THE MQTT PROTOCOL

MQTT is a lightweight event and message-oriented protocol that enables resource-constrained devices to asynchronously communicate and distribute telemetry information to multiple devices efficiently across constrained networks.

As described in the official website [3]:

> "*MQTT stands for MQ Telemetry Transport. It is a publish/subscribe, extremely simple and lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency, or unreliable networks. The design principles are to minimize network bandwidth and device resource requirements whilst also attempting to ensure reliability and some degree of assurance of delivery. These principles also turn out to make the protocol ideal of the emerging "machine-to-machine" (M2M) or "Internet of Things" world of connected devices, and for mobile applications where bandwidth and battery power are at a premium".*

MQTT was invented by Dr. Andy Stanford-Clark of IBM, and Arlen Nipper of Arcom (now Eurotech), in 1999. MQTT originally stands for MQ Telemetry Transport, where MQ denotes the IBM MQ (Message Queuing) product line which is used to exchange messages, but MQTT is no longer considered an acronym and is simply the name of the protocol.

1.2 THE MQTT ARCHITECTURE

The MQTT protocol defines two subjects: a client and a broker. An MQTT broker is the server and the devices connected to the broker are the clients. MQTT uses a single TCP/IP port connection from the client to the server. There are standard ports for MQTT to use. TCP/IP port 1883 is reserved with IANA (Internet Assigned Numbers Authority) for MQTT and port 8883 is reserved for MQTT over SSL/TLS.

The protocol uses a simple Publish-Subscribe architecture which facilitates many-to-one as well as one-to-many distribution. When a client/device wants to send data to one or many other devices, it would assign the data to a specific topic and send it to the broker – it is called publishing on the topic. All the clients that want to receive the message would ask the broker to send any message that gets published on that topic – it is called subscribing to the topic. The broker queues all the received messages, filters them based on the topics, and routes them to the clients that have subscribed to that topic. Figure 2.1 depicts the MQTT architecture with a simple example [2].

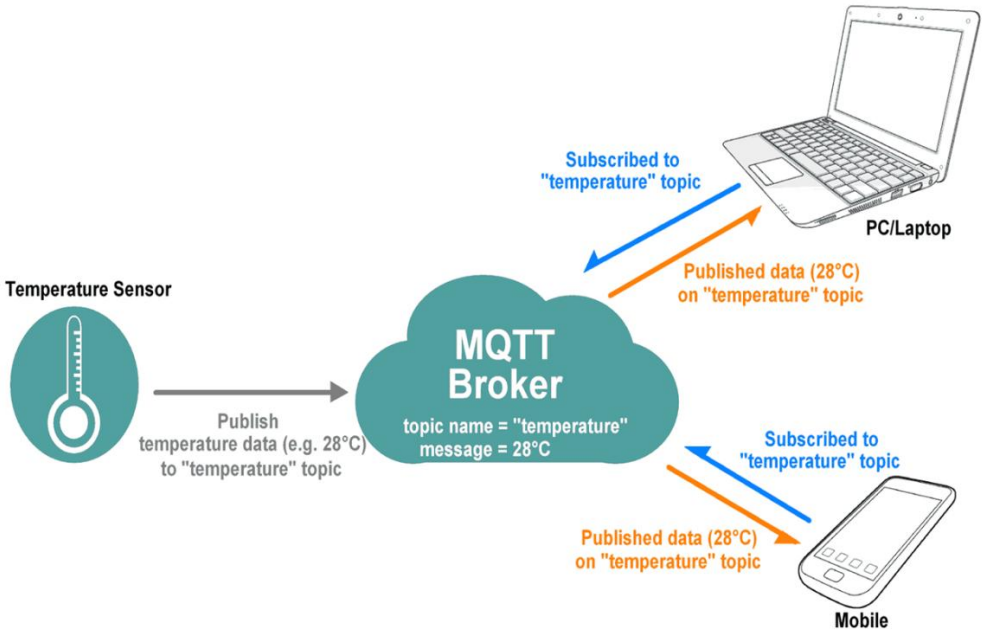


Figure 2.1 The MQTT Publish-Subscribe Architecture

The main components of the MQTT architecture are summarised below:

1. **Message:** Message is the data that gets exchanged between the Publisher-Broker-Subscriber.
2. **Topic:** Topic refers to a UTF-8 string that the broker uses to filter messages for each connected client. Each message is published (sent to the server/broker) on a topic.
3. **Subscriber:** The subscriber subscribes to topics. Any message that is published on the subscribed topic will be received by the subscriber.
4. **Publisher:** Publisher is the one that generates the message. It generates the message and assigns it to a topic and then sends it to the server/broker.
5. **Broker:** The broker is the server that buffers data and pushes them to the subscribers according to the topics to which they have subscribed.

1.3 MQTT TOPICS

MQTT topics are a form of addressing that allows MQTT clients to share information. They are structured in a hierarchy, using the forward-slash (/) as a delimiter. When subscribing to multiple topics two wildcard characters, namely, the # (hash character) – multi-level wildcard and the + (plus character) – single-level wildcard can be used. Some examples of valid topics are house/room1/main-light, house/room1/alarm, house/#, house/+/main-light etc. All topics are dynamically created by a subscribing or publishing client and are not permanent.

1.4 HOW MQTT WORKS

The MQTT protocol works by exchanging a series of MQTT control packets in a defined way. An MQTT session consists of four stages: connection, authentication, communication, and termination just like any other communication protocol.

The clients start by establishing the TCP/IP connection to the broker. Once the TCP connection is established, the client sends an MQTT CONNECT packet with a unique client Identifier, username, and password. The broker authenticates the client using the username and password field in the connect packet. After a successful connection, the client can publish messages on topics using the PUBLISH packet or subscribe to topics to receive messages published on them, using the SUBSCRIBE packet. The MQTT broker sends a PUBLISH packet to the client when it receives a message on the topic subscribed by the client. The broker can set restrictions on the topics a client is allowed to subscribe or publish based on its client ID, username, and password. This makes sure that not any device connected to the broker can publish messages or receive the messages published on a certain topic. The client must send PINGREQ (ping request) packets to the broker periodically to keep the connection alive. Finally, the connection can be terminated by sending an MQTT DISCONNECT packet from the client side. The broker can also terminate the session in case the keep-alive period expires or the client violates the set restrictions or behaves abnormally. The broker responds to the packets from a client with the corresponding acknowledgment packets.

The MQTT protocol provides Quality of service (QoS) levels for reliability. The QoS level determines how each MQTT message is delivered and must be specified for every message published. The following are the available QoS levels with which a message can be published.

1. **QoS 0 (Fire and forget/At most once):** The client or the broker publishes a packet and does not wait for an ACK message from the other end to ensure message delivery. Messages are delivered according to the best efforts of the operating environment. Message loss can occur.

2. **QoS 1 (At least once):** The client or the broker waits for the PUBACK packet from the other end to ensure that the message is delivered at least once. Messages are assured to arrive but duplicates can occur.

3. **QoS 2 (Exactly once):** The client and the broker exchange a set of three more control packets following a PUBLISH packet which are PUBREC (publish received), PUBREL (publish release), and PUBCOMP (publish complete). Messages are assured to be delivered exactly once.

The client that publishes the message to the broker defines the QoS level of the message when it sends the message to the broker. The broker transmits this message to subscribing clients using the QoS level that each subscribing client defines during the subscription process. If the subscribing client defines a lower QoS than the publishing client, the broker transmits the message with the lower quality of service. Higher levels of QoS provide reliability but at the same time increase the latency and bandwidth requirements. Hence one must carefully choose the QoS levels based on the application. Figure 2.2 shows the exchange of Mqtt control packets between the clients and the MQTT broker in an MQTT Network.

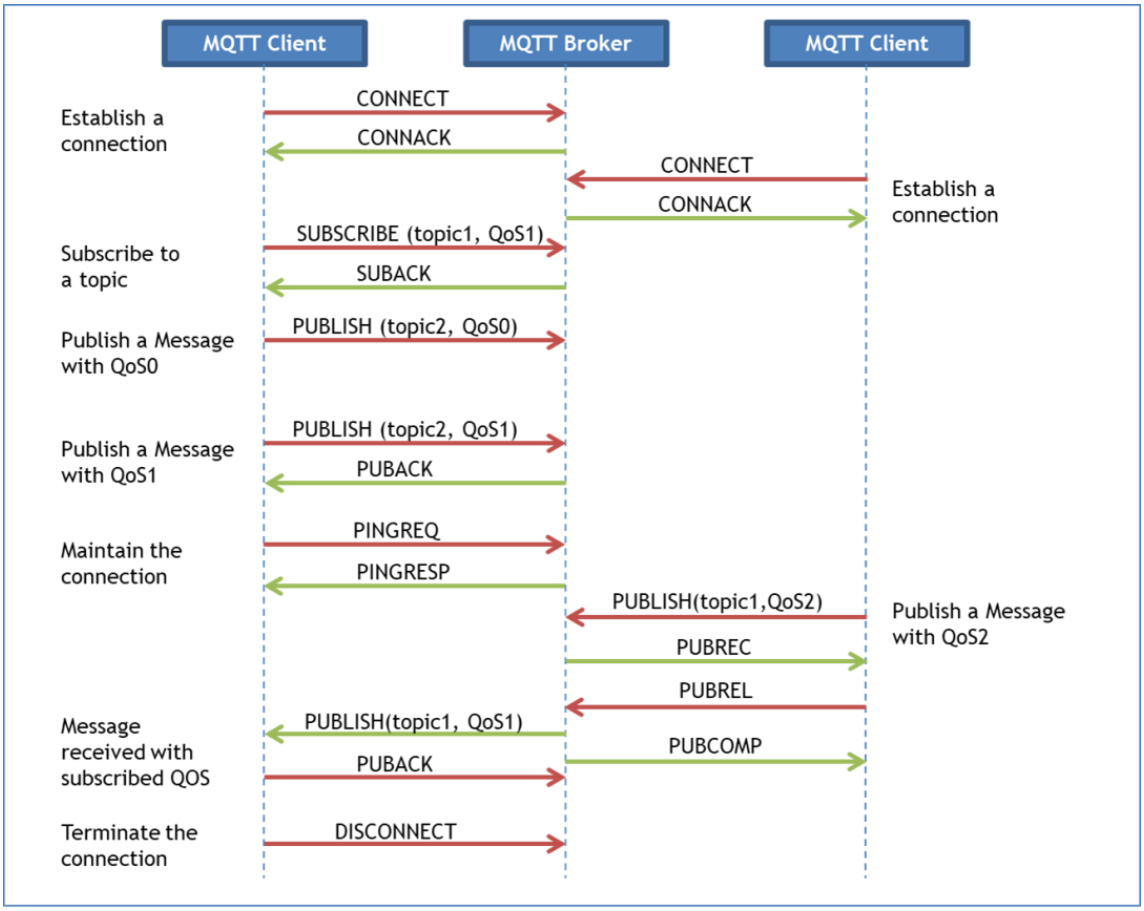


Figure 2.2 Exchange of control packets in an MQTT Network

1.5 OTHER FEATURES OF THE MQTT PROTOCOL

The MQTT protocol supports many other interesting features that prove to be very helpful in the IoT environment. Some of them available in the MQTT v3.1.1 are described below.

Last Will and Testament (LWT): The last will and testament message is used to notify subscribers of an unexpected shut down of the publisher. When an MQTT client connects to the MQTT server it can define a topic and a message that needs to be published automatically on that topic when it unexpectedly disconnects. This is also called the *"Last will and testament" (LWT)*. When the client unexpectedly disconnects,

the keep-alive timer at the server-side detects that the client has not sent any message or the keep alive PINGREQ. Hence the server immediately publishes the Will message on the Will topic specified by the client.

Retained Messages: An MQTT message can be published with the *retained* flag in the publish packet set to true. The broker stores the last retained message and the corresponding QoS for each topic. Each client that subscribes to a topic pattern that matches the topic of the retained message receives the retained message immediately after they subscribe. This helps the newly-subscribed clients to get a status update immediately after they subscribe to a topic.

Persistent Session: If the connection between the client and broker is interrupted during a non-persistent session, these topics are lost and the client needs to subscribe again on reconnect. To avoid this problem, the client can request a persistent session by setting the *cleanSession* flag in the connect packet to false when it connects to the broker. In a persistent session, the broker stores the required information to resume the session without a problem when a disconnected client reconnects.

MQTT v5.0 update, which is a successor of v3.1.1, adds several features like better error reporting, including metadata in message headers, shared subscriptions for client load balancing, message expiry option, session expiry to improve session management, zero-length publish topic and topic aliasing to reduce bandwidth usage, etc.

1.6 SECURITY IN MQTT

1. Identity management and Authentication:

The MQTT provides some client Identity management and authentication features to provide some basic client connection security. All the clients must provide a unique client ID when establishing a new connection. Some broker implementations allow imposing client id prefix restrictions on the client name to have control on the devices getting connected. Optionally, a username and password can be used to authenticate the clients and to restrict access to the broker.

2. Authorization:

The MQTT broker can be configured to allow or restrict a user from performing certain actions so that different users can be authorized to perform different actions. Restrictions on publishing or subscribing to topics can be imposed based on the Client ID or the username. Role-Based Access Controls (RBAC) and Access Control List (ACL) are two of the most common types of authorization used. Using ACL or RBAC a broker can be configured with topic permissions. During the run-time, the broker can determine allowed topics, allowed operations, and allowed quality of service. If a client attempts to perform an unauthorized operation, the broker can perform actions such as disconnecting from the client or acknowledging the client with a failure code.

3. Encryption:

The protocol relies on TCP as the transport protocol and does not have security and encryption built into it. Though it provides an optional username and password for client authentication, it is sent in cleartext. This leaves a chance for Man-in-the-middle (MITM) attacks to be executed to steal the credentials. To avoid this, MQTT brokers allow

encryption using SSL/TLS (Transport Layer Security) instead of plain TCP, but this is not integrated into the protocol to keep it lightweight.

Security using TLS comes at a cost in terms of CPU usage and communication overhead. The additional CPU usage can be a problem for constrained devices that are not designed for performing computationally intensive tasks. Also, the communication overhead of the TLS handshake can be significant if the MQTT client connections are expected to be short-lived. Techniques such as *Session Resumption* can improve TLS performance in such cases. Since every packet is encrypted in TLS, packets transmitted have additional overhead compared to the unencrypted packets. Additionally, TLS has a large code footprint and increases memory usage significantly.

Payload encryption is another, not very popular method to secure the data on the line. This is done in the application level to encrypt only the payload transmitted and hence does not encrypt the username and password. The data is encrypted end to end between the publishing device and the subscribing device. This means that both the subscriber and the publisher must use same the encryption with a fixed pre-shared encryption key. The need to generate the encryption key manually and hardcoding it in all the clients makes this method less secure despite the additional complexity.

1.7 COMPARISON WITH OTHER COMPETING IOT PROTOCOLS

There are many other message/telemetry transfer protocols that compete with Mqtt in the IoT environment. Some of them are discussed here.

Constrained Application Protocol (CoAP) is primarily, a one to one client-server protocol developed for IoT applications that run over UDP. In contrast with the data-centric publish/subscribe communication used in Mqtt, this protocol is designed to interoperate with HTTP and hence is document-centric and uses a request/response communication pattern. DTLS, Datagram Transport Layer Security can be employed to provide security. Recent updates have added support for a publish/subscribe architecture that runs over TCP like MQTT.

Advanced Message Queuing Protocol (AMQP) is another common IoT protocol that runs over TCP and supports both request/response and publish/subscribe communication patterns. Transport Layer Security (TLS) and Simple Authentication and Security Layer (SASL) are a part of the protocol and hence it provides better security options. But it is not the best option for constrained devices and constrained networks since it is designed with more advanced features and has more communication overhead than MQTT. Moreover, it does not provide options for reliability and uses only the fire and forget policy.

Extensible Messaging and Presence Protocol (XMPP) is a protocol originally designed for instant messaging (IM), presence, multi-party chat, voice, and video calls. The XMPP-IoT is a variant that has added extensions for use in IoT. It is XML based and hence requires an XML parser that increases the memory footprint. XMPP supports different communication patterns like Request/Response, Asynchronous Messaging, and Publish/Subscribe. Similar to AMQP, it uses TLS and SASL to provide security and does

not provide QoS levels for reliability. XMPP is basically an IM protocol and has a much higher overhead when compared to MQTT.

Data Distribution Service (DDS) was designed specifically to address machine-to-machine (M2M) communication and uses a publish/subscribe pattern like MQTT. It also provides multiple QoS levels for reliability. The main difference between the two is that DDS typically uses UDP and is decentralized, i.e. the messages are not routed through a centralized server, and instead peer to peer communication is used. Unicast messaging is used for one-to-one communication and Multicast messaging is used for distribution from one to many.

All of these protocols have their pros and cons. One must choose the protocol that best suits their application and the device and network constraints. When considering applications like remote sensing and control, real-time data analytics, and data monitoring, MQTT proves to the best option since it is designed specifically for such use cases where the memory, power, and bandwidth usage are constrained.

1.8 PROS AND CONS OF MQTT

MQTT is an extremely simple and lightweight messaging protocol, with a publish/subscribe architecture. It is designed to be easy and straightforward to deploy with a small code footprint and is capable of supporting thousands of clients with a single server. The protocol is data-agnostic, i.e. it is possible to send images, text in any encoding, encrypted data, and virtually every data in binary. The provision of different QoS levels enables the user to choose between reliability and performance based on the application requirement. The size of the data packets is minimized to reduce the network as well as the power usage, which is of utmost importance in IoT devices. In addition, Mqtt is fast and efficient in data transmission and delivery even in congested or unreliable network connections.

The protocol also has some disadvantages in its current implementation. As discussed previously, MQTT has minimal authentication features built into the protocol. Usernames and passwords are sent in clear text and for encryption and security, one must employ SSL/TLS, which is not lightweight and increases the code footprint. Even with TLS, issuing, and managing client certificates for client authentication is a complex process due to a large number of IoT devices. Another major challenge is that MQTT's topic structure makes it difficult to scale the MQTT network to a global level. As the number of connected devices increases, the complexity on the broker side increases which in turn affects the overall performance.

2. OVERVIEW AND PROBLEM STATEMENT

There are many open-source implementations of the MQTT protocol that can be used to set up an MQTT broker or a client. While most of them are for operating systems like Linux, Windows, and Mac, there are very few implementations available for embedded systems which are a major part of the IoT world. Eclipse Paho Embedded C library is one very popular C based library that is widely used in embedded IoT applications.

The Eclipse Paho Embedded C Library

Eclipse Paho Embedded C library is a lightweight, open-source MQTT v3.1.1 implementation for embedded devices. It is a generic implementation that is not reliant on any particular libraries for networking, threading, or memory management. This is done to make the library portable since TCP/IP stacks and multithreading libraries are not standardized. Complete implementation of the MQTT protocol using the Eclipse Paho library in any embedded platform requires the implementation of network calls, timer functions, and memory management functions. Threading functions must be implemented if one wants the MQTT communication to function on a separate thread. Besides, security implementation requires integrating a TLS library and implementing the necessary functions.

Problem Statement

Using the Eclipse Paho Embedded C library, write an abstract, easy to use wrapper MQTT library for STM32 Microcontroller devices. Implement all the necessary functions to provide a clean and complete MQTT client library along with TLS deployment for security.

3. IMPLEMENTATION AND TESTING

The MQTT library was written and compiled in the IAR Embedded Workbench for ARM which is an IDE for programming embedded applications. Two different versions were written, **MQTT-LWIP Client library** for use in devices that use Ethernet for communication and the **MQTT-GSM Client library** for devices that use a 2G GSM module for data communication.

MQTT-LWIP Client library:

- For communication using Ethernet at the physical layer.
- Uses the LWIP (Lightweight IP) TCP/IP stack for networking.
- Uses the mbedTLS library for security.

MQTT-GSM Client library:

- For communication using the Neoway N10 2G GSM module.
- Uses the TCP stack and the TLS stack from the GSM module for networking and security.

The library provides a simple set of functions making it easier and quicker for the user to set up an MQTT client for their application. Table 2.1 shows the API's provided along with their functionalities. The following functionalities were implemented to simplify the data exchange in the remote data monitoring and control process. Figure 2.3 illustrates the usage of these functionalities.

1. **NOTIFY:** The embedded device can send data at any time over the open MQTT connection using this functionality. The receiving client, which can be a central management server (CMS) or a mobile app, can optionally send back an acknowledgment.

2. **GET Request/Response:** The CMS or the mobile app can request one or more devices to send data at any time. The IoT devices can then send the requested data.
3. **SET Request/Response:** The CMS or the mobile app can request one or more devices to set a certain value to any defined parameter at any time. The IoT devices set the values accordingly and can optionally send back a response.

The MQTT-LWIP Client library was tested on the STM32f207 microcontroller board using a simple LED application. An LED light on the microcontroller board was monitored and controlled from a mobile application. The MQTT-GSM Client library was tested on the Inverterless 500 system, which uses the STM32f072 microcontroller board. All the data that was previously communicated to the server using HTTP, was published using MQTT.

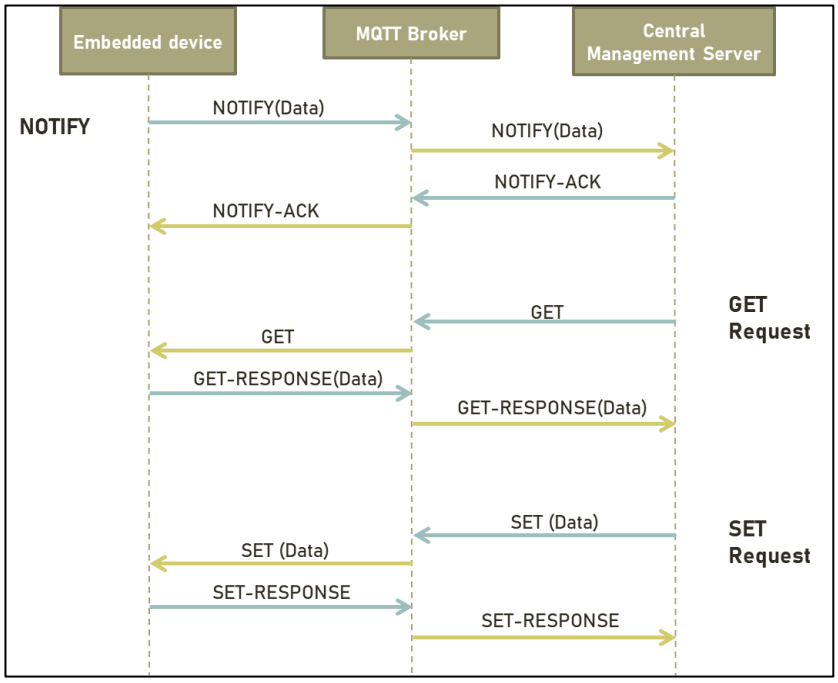


Figure 2.3 Data Exchange functionalities provided

Table 2.1 API's provided and their functionalities

API	Description
int **MqttInit**(char* ip_addr, char* port, char* clientID, char* username, char* password, char useTLS, char* cacert)	• Establishes TCP connection • Creates an MQTT session
int **Notify**(void* data, size_t len)	• Publishes the data
int **MqttYieldMessage**()	• Polls for incoming messages for 150ms • Pings the server if KeepAlive expires • KeepAlive set as 5mins
void **NotifyAckCb**(void* payload, size_t len) void **GetReqCb**(void* payload, size_t len) void **SetReqCb**(void* payload, size_t len)	• Client can receive NotifyACK , a GET request or a SET request • These message received callbacks must be implemented by the user
int **SendGetResp**(void* data, size_t len) int **SendSetResp**(void* data, size_t len)	• Sends response to the GET/SET request received
int **MqttDisconnect**()	• Disconnects from MQTT broker • Closes TCP session

REFERENCES

[1] Anusha Ramachandran, Sairam Mannar, & Ashok Jhunjhunwala. (2016). Inverterless Solar-DC System Design for OffGrid and Near Off-Grid Indian Homes. IEEE Transaction.

[2] ElectronicWings. (n.d.). Retrieved June 10, 2020, from https://www.electronicwings.com/nodemcu/nodemcu-mqtt-client-with-esplorer-ide

[3] MQTT. (n.d.). Retrieved June 10, 2020, from http://mqtt.org/documentation