HIERARCHICAL DECENTRALIZED CONTROL TECHNIQUES OF A MODEL DC MICROGRID

by

MARC A. CARBONE

Submitted in partial fulfillment of the requirements
For the degree of Master of Science

Thesis Advisor: Dr. Kenneth Loparo

Department of Electrical Engineering and Computer Science
CASE WESTERN RESERVE UNIVERSITY

AUGUST 2016
CASE WESTERN RESERVE UNIVERSITY

SCHOOL OF GRADUATE STUDIES

We hereby approve the thesis of

Marc A. Carbone

Candidate for the degree of Master of Science

Committee Chair

Professor Kenneth A. Loparo, Ph.D.

Committee Member

Professor Mingguo Hong, Ph.D.

Committee Member

Professor Marc Buchner, Ph.D.

Date of Defense

23 May 2015
Dedicated to the memory of Martin J. Kirwan
Table of Contents

Table of Contents ........................................................................................................ iv
List of Tables .................................................................................................................. vi
List of Figures .................................................................................................................. vii
Acknowledgements ........................................................................................................ ix
Abstract ........................................................................................................................... x

Chapter 1: Introduction .................................................................................................. 1
1.1 Motivation ................................................................................................................... 1
1.2 Literature Review ..................................................................................................... 2
1.3 System Set-up .......................................................................................................... 4
  1.3.1 Hardware Model ................................................................................................... 4
  1.3.2 Control Overview ............................................................................................... 5
  1.3.3 Simulation Techniques ....................................................................................... 7

Chapter 2: Control ....................................................................................................... 10
2.1 Control Methodology ............................................................................................... 10
2.2 Control Hierarchy .................................................................................................... 12
2.3 Network Disagreements ......................................................................................... 15
  2.3.1 Iterated Prisoner’s Dilemma ............................................................................... 16
  2.3.2 The Pavlov Strategy .......................................................................................... 17
2.4 Consensus Methods ............................................................................................... 19
  2.4.1 The Paxos Algorithm ......................................................................................... 20
  2.4.2 Byzantine Fault Tolerance ................................................................................. 23
  2.4.3 FAB Paxos .......................................................................................................... 24

Chapter 3: DC Microgrid Simulation .......................................................................... 28
3.1 Paxos Simulation ...................................................................................................... 28
3.2 JADE ......................................................................................................................... 30
3.2.1 Implementing Control Methods Within JADE .................................................. 32
3.3 Connecting the Java/Matlab Controller ............................................................ 34
3.4 Simulation Results ......................................................................................... 36
3.4.1 Load Shedding ....................................................................................... 39
3.4.2 JADE Simulation .................................................................................... 46

Chapter 4: Hardware Demonstration Results ....................................................... 47
4.1 Communication Overview ............................................................................ 47
4.2 Hardware Demonstration ............................................................................ 48

Chapter 5: Conclusions and Future Work ............................................................. 53
5.1 Conclusions ............................................................................................... 53
5.2 Future Work ............................................................................................. 54
Bibliography ........................................................................................................ 57
List of Tables

Table 1: Representation of the Iterated Prisoner’s Dilemma game ........................................ 16

Table 2a: SOC results of batteries a and b for the Paxos controller ........................................ 37

Table 2b: SOC results of batteries a and b for the decentralized controller ............................... 37

Table 3: Load Shedding for Stage 2 of Resource Allocation .................................................. 45
List of Figures

Figure 1: Common dc Microgrid configuration 2

Figure 2: DC Microgrid Schematic 5

Figure 3: Network Topology Algorithm 12

Figure 4: Visualization of the Clearing Cost Function 13

Figure 5: Resource Allocation Algorithm 14

Figure 6: Battery SOC using the Pavlov method 18

Figure 7: Battery SOC using the Paxos algorithm 29

Figure 8: Communication flow of Network Topology algorithm in JADE 33

Figure 9a: Battery SOC using the decentralized controller with random priorities 36

Figure 9b: Battery SOC using the Paxos controller with random priorities 36

Figure 10: Battery SOC without load shedding 40

Figure 11: Battery SOC with load shedding 42

Figure 12 a-d: Load Shedding for Stage 1 Resource Allocation 43

Figure 13 a: SOC for Java/Matlab Simulation 46

Figure 13 b: Configuration Selection (1-5) of the Network 46

Figure 14 (a-f): Hardware Demonstration Results 50
Figure 15: Current Measured from Sources a and b

Figure 16: Beta Version of the DCMG GUI
Acknowledgements

Firstly I would like to thank my advisor, Professor Kenneth A. Loparo, for his guidance and encouragement throughout this process. Without his support I would not have had the opportunity to join this program. Also, I thank Ryan May for his advice and support since the beginning of this project. I am grateful for his patience while working with me on this project, and for sharing his expertise with me. Also I would like to thank him for the use of his socket functions for the Java/Matlab controller. I am grateful for Jordan Murray for his hours of work on the design and setup of the hardware for this project. His tireless efforts made the final phase of this possible, and I am very appreciative of all of his advice for putting this project together. I thank Tim Dever for his perspectives and the GUI beta for the DCMG. Lastly I would like to express my gratitude to my parents for the endless amount of care and support I have received in my lifetime.
Hierarchical Decentralized Control Techniques of a Model DC Microgrid

Abstract

by

MARC A. CARBONE

Abstract

As the practicality of decentralized generation progresses, the control and mitigation techniques of the large centralized energy grid become unsuitable solutions for power distribution. The creation of autonomous control on a dc microgrid is necessary to manage the system objectives of the future power grid. In order to begin this study, control algorithms must be tested in both software and hardware experiments. This research will explore the control efficacy of supporting the system objectives on a lab-scale dc microgrid, using a hierarchical control algorithm. The algorithm will determine the network topology of the system, as well as distribute power to the loads in order of priority.
1.1 Motivation

Recent advancements in generation technology have inspired great interest in the field of direct current grid operating systems [1]. Many modern renewable energy sources come in small capacity, and often in dc. There is strong evidence that these distributed generation (DG) techniques are growing rapidly. For example, total solar power generation grew by 6.2 GW in 2014 in the U.S. This is a 30 percent increase over the total generation in 2013, and represents nearly $18 billion dollars in new investments according to the Solar Energy Industries Association. Other distributed power generation techniques, such as wind, fuel cells, and gas cogenerators have seen similar rises in production in recent years. The Department of Energy estimates that by the year 2050, the total wind capacity across the United States will be over 400 GW, roughly 35% of the national end-use electricity demand [2]. All of these new generation technologies are beneficial for diversifying our energy resources. However, when distributed generators are connected to utility grids, they cause problems such as voltage rise and protection malfunctions [3]. Not to mention, an increasing number of consumer loads, like computers and LED lights require dc current, which increases the practicality of dc networks.

Interest in dc systems has been sparked by the increased development of power electronics technology, greater interest in renewable energy sources, and the improvement and cost reduction of energy storage systems. Decentralized network
distribution systems will be essential in the implementation of these types of local
generation technologies. Localized generation on a mass scale causes the large
centralized ac systems, which currently power almost all of the United States to be a less
than optimal solution for power distribution. Due to the majority of infrastructure already
designed for ac current, a majority of microgrid research has been done using ac systems
[4]. With the security of electrical power systems becoming more important in our daily
lives, the proper integration of DGs becomes crucial. The switch to dc systems must
maintain and even improve upon the protection standards used in todays ac systems.
Thus there is a demand for research in the field of decentralized hierarchical control of
the dc microgrid.

1.2 Literature Review

Most research in the control of the dc microgrid focuses on operation during grid
connected mode [5]. For example, Zhang, Wu, Xing, and Sun’s study, *Power Control of
a dc Microgrid Using dc Bus Signaling* tests the grid in four operating modes [6]. Two of
these modes are islanded, which means the grid does not receive any power from the
main grid and must supply its loads based on its own generation. In their study, one
mode uses the main battery to supply power to the grid for a short period of time. The
other supplies dc bus voltage regulation by photovoltaic (PV) sources. Although the
islanding mode is taken into account in this study, neither of these modes provides
solutions for long-term self-sustaining dc

![Figure 1 - Common dc Microgrid Configuration](image-url)
Lopes, Moreira, and Madureira’s study, *Defining Control Strategies for MG Islanded Operation*, examines two operating modes, one of which is islanded in an Emergency Mode [7]. In this situation a fault in the system has occurred, such as an open transmission line, or certain maintenance requirements on the network. Again, this mode fails to deliver sustainable operation for islanded dc microgrids.

In addition, these studies do not support dynamic network topology changes to manage the system requirements. Figure 1 shows a schematic similar to Li, Yan, and Kai’s microgrid where the local loads have the ability to receive power from the main ac grid, a series of photovoltaic sources, or a battery storage system[6]. However, all of the loads must get power from the same source, which limits the systems ability to reroute power in the event of a fault or failure. Another shortcoming of this design is that there is only one battery storage location in the network. Since residential and commercial generation will be a common feature of the future grid, power storage will be a necessary tool for making use of locally generated power. Battery technology has improved greatly in recent years [8]. Companies in 2015 are preparing both residential and commercial grade battery storage systems for local generation use on a large scale. These batteries are capable of producing 10kWh and 100kWh respectively. If more power is required, the residential batteries can be stacked to create up to 90kWh for nine batteries in one home. The commercial batteries can be scaled greatly, upwards of 1GWh [9]. As battery and generation technology improves, understanding the effects of energy storage systems on the grid is necessary. Therefore, in order to conduct a meaningful study on the implementation of grid storage, our study will observe multiple battery storage systems.
attached to local power sources to study the additional control abilities in our simulated dc microgrid.

1.3 System Set-up

It is evident that the topology and operating requirements of a dc microgrid can vary greatly. As a result, different microgrid studies need to be conducted in order to understand the range of capabilities of the subject. Our study will explore the grid capabilities only in islanding mode. Operating in islanding mode demonstrates how well the microgrid can function in emergency situations. For example, if there is a line fault or insufficient supply from the main grid. This will provide valuable information about how a dc microgrid responds without support from outside generation sources, and test the system’s stability locally. Also, as local generation technology increases, it is plausible that microgrids will become self-sustaining and only require generation from the main grid in times of emergency. For these reasons there is great need to investigate the control techniques of the dc microgrid in a stand-alone setting.

1.3.1 Hardware Model

In order to collect meaningful data on this subject, we will run simulations on a lab-scale dc microgrid. This will allow us to test multiple control techniques on actual hardware, and provide insight on how to implement these techniques on a real microgrid. The grid is set up in a loop (Figure 2), witch allows the switches to change the network topology based on the systems needs. This configuration is unique in that the loads can receive power from different sources depending on the topology state of the grid, which provides additional flexibility when managing the system. The loads are placed on four
busses around the loop, where busses 1 and 2 are considered residential (smaller load size), and busses 3 and 4 are commercial (larger load size). Each bus has three loads connected to it, as well as an input for a distributed generation source. Our study will address the situation of two generators, one attached to bus 1 and the other on bus 3. Voltage and current sensors are located around the grid, and will help determine the state at various locations throughout the network. There are also several scenario generator locations in the system, which have the ability to cause a fault in the network, unknown to the system controller.

1.3.2 Control Overview

Because the motivation for dc microgrid research is largely based on distributed ‘behind the meter’ generation, a centralized control technique will prove insufficient. To create a communication infrastructure from localized generation sources to a centralized
controller would become incredibly complex on a large scale. Instead an autonomous decentralized control algorithm using agent-based communication will be necessary for the dc network. This will allow elements in the system to talk to each other and make decisions about how to manage the system remotely. This approach is advantageous because it demonstrates greater reliability in the event of a fault or failure in the system, and improves response techniques. However, the management of the system objectives can be less optimal than centralized methods due to the limited information available at each point in the network.

This study will investigate two decentralized control strategies. The first uses a series of priority-based auctions to determine the network topology. For each iteration, all of the generation sources attempt to distribute power to the loads. In the first auction, every power source makes an offer to each bus, based on the amount of available power. Then the busses choose to purchase power from the least expensive bus. Note that battery storage systems can behave as sources or loads based on their state of charge (SOC). In order to avoid all of the loads moving to the more available power source, there is a switching penalty added to the cost of a power source, when that bus is not connected to that power source in the previous iteration. The switching penalty is based on priority so that the most important loads do not risk losing power by moving to a cheaper power provider. This also helps minimize switching in the network.

The second control algorithm uses a Paxos based consensus algorithm to determine a leader among the proposer agents (bus agents). The leader is determined based on the priority of the bus agents during the current iteration. All of the proposers send their vote to the leader for their preferred network topology, using the same auction
method described in the first algorithm. Once all of the votes are received, the leader makes a decision as to how the network will be configured. The topology selection is made once a majority decision has been made. Before the next iteration begins, the algorithm repeats to prepare for the upcoming iteration.

Both techniques then use the same resource allocation algorithm to determine how power will be distributed, and which low priority loads will be shed in the event of a power shortage. Finally power is distributed to the loads, and the battery SOC, generation schedule, and priority are all updated to repeat the algorithm, for the next iteration. Both of these control techniques are designed to balance the battery storage levels as well as minimize the number of switch states for each planning period. The control architecture also ensures that higher priority loads are served first.

1.3.3 Simulation Techniques

First, the control algorithms and dc microgrid will be modeled in Matlab software. The first stage of testing allows us to easily monitor the strengths and weaknesses of a control strategy, as well as simplifying the communication process between agents for faster results on the effectiveness of the control strategy. Matlab performs well in extracting meaningful data from the algorithm and exporting it in a usable format (e.g. plots, flow charts, etc.). The software simulation results will allow us to compare control strategies, as well as test the network limits to ensure none of the hardware will be damaged in real testing. It is important to ensure that we do not produce any over currents that would exceed the equipment ratings of the hardware and damage the
network. Lastly, the Matlab code will provide a framework in which we compare our JADE and hardware experiment results.

The second stage of testing will run the agents in JADE software, as an intermediate step towards hardware integration. JADE provides useful execution in peer-to-peer asynchronous agent communication, set up in the Java language [10]. Here, the agents will run their control algorithms in JADE, and send the results back to Matlab through TCP sockets. Then the Matlab program will update the input conditions such as, battery SOC, generator availability, and network configuration, and return the new inputs to the agents in JADE. This allows us to make sure the JADE agents are communicating correctly as well as test that the TCP socket communication is successful. Once we are able to confirm that the results from the JADE simulation match the results from the Matlab simulation we can begin the hardware integration stage.

The third and final stage of this study will test the JADE agents on the actual hardware of the dc microgrid. In the hardware loop, the system input information (battery SOC, switch state, etc.) is collected using the network sensors. That information is shared with Matlab through a few wrapper functions in a CIP stack. Matlab sends the data to the JADE agents over TCP sockets, which processes the data in its control algorithm. The results of the network topology algorithm are sent back to Matlab through the TCP sockets and Matlab uses the JADE results to process the resource allocation algorithm. Next, Matlab shares the information to the PLC through the wrapper functions, where the controllable elements on the microgrid make the necessary changes. Finally, once the results of the hardware demonstration are confirmed to match the software results a balanced distributed dc microgrid control system is achieved.
After the controller is in place, we are able to test the robustness of the controller, by creating different contingency scenarios, in which a dc microgrid may actually face. This provides vital information about the performance of the automation of each controller when confronted with real world failures in the network. From this, we can compare the strengths and weaknesses of each controller technique. Testing several scenarios in the software loop allows us to expose limitations in the controllers and make the necessary changes to the algorithm before testing them on real hardware.

Overall, there is a large amount of dc microgrid research and testing that must be completed before renewable DGs like solar and fuel cells can be introduced to the grid on a massive scale. Localized generation is best suited for smaller grid networks like the dc microgrid. In order to make distributed generation a possibility, the effects of local generation combined with multiple battery storage systems must be studied in great detail. Our research examines a dc microgrid system operating only in islanding mode, and addresses the aforementioned concerns using autonomous hierarchical control techniques. More extensive and detailed simulations will be necessary before introducing mass localized generation into a real dc microgrid.
Chapter 2: Control

2.1 Control Methodology

To create an effective controller, a decentralized strategy must be developed to manage all of the system objectives. These objectives include: maintaining the state of charge (SOC) of both generation storage devices, distributing power to loads in order of priority, and minimizing the amount of switch states in the system. According to Zhang, “The key point of power management in dc microgrids is to maintain the power balance between energy sources, storage devices, and loads at any time”[11]. Thus balancing the power balance will be our highest priority. In order to have a reliable network, the storage at both ends need to maintain some amount of operating reserve in the event of a power shortage. Also, the storage devices are limited to a 25-90% charge range to avoid damage (for lead-acid batteries).

Our second objective ensures that the most important loads are served first. For each time interval, each load is assigned a unique priority in the set of real numbers between 0 and 1. The control algorithm will distribute load using the priority scheme. This assures that the crucial loads remain on in events where load shedding may occur. For example, a hospital may yield a higher priority than a residential home. Then in the event of an emergency power shortage, the controller will guarantee that the hospital receives power before the residential home.

Lastly, switching can be costly to the system as well as potentially causing faults. Limiting the amount of switches in the network will increase the lifespan of the switching
components and reduce the amount of maintenance required in the system. Therefore we will attempt to minimize the amount of network topology switches. However, this is the third and least important objective, and the control methodology will meet the standards of the first two objectives before minimizing the amount of switch states.

Again, the system in consideration is shown in Figure 2. There are four ‘streets’ or busses where loads are located. Two of the streets are considered residential, and the other two are commercial. A switch in increments of 3W sets the residential loads, where the loads can range from 0W to 15W. The commercial loads range from 0W to 25W (in increments of 5W). There are several sectionalizers and crossties located throughout the system to change the network topology. These will act as our system switches. The system has the ability to connect generation elements at a main input to the left of the system, or at any of the commercial or residential streets. For our purposes, we will study the case of two generation sources at bus 1 and bus 3. Again, this will serve as our model for the use of localized generation. In the future, the main source connection on the left of the schematic will be tied to the labs ac microgrid where power can be distributed from one grid to the other. However, grid interoperability is beyond the scope of study.

The dc Microgrid has voltage sensors (orange-yellow) at each of the streets and the main source connection point. This will be useful for load shedding during the resource allocation algorithm discussed later in this paper. Current sensors (blue) are located on every line in the distribution network, and before each load. Lastly, located throughout the system are the fault actuators (green). These components are able to force
a fault in the circuit regardless of the scenario generator schedule, and unknown to the system controller.

2.2 Control Hierarchy

To achieve the system objectives, a hierarchical control algorithm is developed.

- **A. Network Topology**
  - Step 1: Given initial battery SOC and $C_{nom} = 3$ determine $C_1$ and $C_2$.
    \[ C_1 - C_2 = -5 \cdot \Delta SOC \]
  - Step 2: Assign priority to each PDU.
  - Step 3: Calculate switching penalty for each load.
    \[ Pen_i = \frac{L_i}{1000} + 0.05 \]
    - Where $L_i$ is the value of the largest load (in kW) on bus $i$.
  - Step 4: Calculate the total cost of each PDU to purchase power from Channel 1 and Channel 2.
    \[ CP1 = C_1 \quad \text{or} \quad CP2 = C_2 + Pen_i \]
    - This shows the relative cost for a PDU already located on Channel 1.
  - Step 5: Determine the new location of each PDU by selecting the Channel with the lower cost.

![Figure 3 - Network Topology Algorithm](image)

The layers in the algorithm are set up in a manner similar to that of the terrestrial Regional Transmission Organizations, and so many of the same objectives on the actual grid may also be met in our lab microgrid.

The first step in the algorithm is to run a Network Topology test (Fig. 3), that is, to determine the switch state configuration of the microgrid. This step is crucial to solving the system objectives because the loads need to be distributed in a manner that will maintain the battery state of charge as well as minimizing the amount of switch states. Below are the five steps to compute the Network Topology of the system.
Note that this method requires no extensive calculations or differential equations, resulting in a low cost algorithm. The gains are set such that the channels are balanced; yet switching states are minimized. Increasing the coefficient on the \( \Delta \text{SOC} \) term in step 1 minimizes the amount of switches, but increases the permitted gap in SOC between the battery storage systems. Therefore, depending on the needs of a system, the correct level of switch states and battery storage limits can be met.

The second layer of the control algorithm is to preform a Resource Allocation. This step determines how much power each street will receive for the upcoming iteration. This process is completed using a clearing cost function, where each street decides on a priority based bid and the amount of power it requires. The power supplying elements will make offers to sell power based on the amount of availability and the cost associated with the type of generation. For example, power from the solar arrays is set at $0.01 in order to maximize their impact while they are in use. The batteries will make both offers and bids, because they have the capability to be both a power supplier and a load (based on their charge/discharge needs). The battery cost to sell is calculated in the Network Topology algorithm and the bid to buy power is set at one-cent-less than the cost to sell power, so battery can never purchase power from itself. This way, as the battery SOC decreases, the cost to sell power will become larger, and the likelihood of the battery charging will become greater.

Thus power balancing is put in effect.

Once all of the bids and offers are

![Figure 4 - Visualization of the Clearing Cost Function](image-url)
collected, they can be modeled as two overlapping step functions. The priority levels are plotted on the y-axis and the power (in kW) is plotted on the x-axis. This creates a graph resembling Figure 3. The clearing cost is calculated as the intersection of these two step functions. Every element to the left of the intersection point will receive power, or partial power if a portion of a bid is to the left of the clearing cost. Conversely, those elements to the right of the clearing value will receive no power during that iteration.

In the algorithm the clearing cost process defines the step functions only by a series of points (the corners of each step) to reduce the cost of the process. Therefore it becomes more difficult to determine where the two functions will intersect. To solve this, the clearing cost algorithm examines the location of each bid point with respect to the offers, and then moves on to the next highest bid and repeats the examination. Depending on the result of each points location, the program will either save the last possible clearing cost value or move on to the next point. After each bid is tested (4 streets and one battery bid) the clearing cost is determined. Displayed below is the algorithm for the Resource Allocation.

This algorithm is repeated for the third and final layer of the control architecture, which is Resource Allocation 2. In this step, the streets have received power and send

<table>
<thead>
<tr>
<th>B. Resource Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>– Step 1: Receive bids from PDU’s and offers from power sources.</td>
</tr>
<tr>
<td>– Step 2: Calculate clearing cost as the intersection of the offers and the bids.</td>
</tr>
<tr>
<td>– Step 3: Distribute power to the PDU’s that have cleared.</td>
</tr>
<tr>
<td>– Step 4: Repeat Steps 1-3 for PDU to load connections for each PDU.</td>
</tr>
<tr>
<td>– Step 5: Update battery SOC and repeat network topology algorithm.</td>
</tr>
</tbody>
</table>

Figure 5 - Resource Allocation Algorithm
offers. Then the loads make bids to their corresponding street. Another set of clearing costs are determined for each street, and the loads receive power. This is the final step in a single iteration of the algorithm.

Once the power is distributed for given iteration, the change in state of charge must be calculated for each battery. In the hardware loop, the SOC of each battery will simply be measured in the planning stage of the algorithm. In the software simulation the SOC is calculated using nothing more than the basic principals of Ohm’s Law.

\[
\Delta C = C[k+1] - C[k] = -I[k] \cdot T \\
I[k] = P[k] / V_{nom} \\
SOC[k+1] = C[k+1] / C_{nom} = SOC[k] + \Delta C / C_{nom}
\]

Where, I is the current absorbed or supplied from the battery and T is time in hours. C_{nom} is the total capacity of the battery, 120Ah. Note, if I[k] > 0 the battery is a current source and C[k+1] < C[k]. If I[k] < 0 the battery is a current sink and C[k+1] > C[k]. Also, if I[k] = 0 then C[k+1] = C[k].

2.3 Network Disagreements

Because the topology of this system is of a ‘loop’ nature, not every street to source combination is physically possible. In events where there is a disagreement on a potential topology a sub-layer control strategy must be introduced. To begin, we will consider the case where streets 2 and 4 have made a disagreement on location thus causing an impossible network configuration (e.x. street 2 would like to connect to source 2 and street 4 would like to connect to source 1). Using techniques derived in the field of game theory we have developed one solution to this dilemma.
2.3.1 Iterated Prisoner’s Dilemma

A common problem in the field of game theory is the prisoner’s dilemma. In this game, two individuals who have committed a crime are apprehended. They are in isolated cells and are interrogated. Each player has an option to confess, or defect from the crime. If they both confess they each receive 3 points. If one player confesses and the other defects, the defector receives a maximum penalty for the crime (0 points), and the confessor is released from custody and receives immunity as a state’s witness (5 points). Finally, if both players defect they each receive only 1 point. The game is represented in Table 1.

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>(3,3)</td>
<td>(5,0)</td>
</tr>
<tr>
<td>D</td>
<td>(0,5)</td>
<td>(1,1)</td>
</tr>
</tbody>
</table>

In the dc microgrid, we will make the analogy that to confess will be to agree to go to the source where power is cheaper, and to defect will be to connect to the more costly source. This makes sense because it would be in both busses best interest to connect to the cheaper bus because it is likely that there is more available power. Thus it would always be in the best interest of the network as a whole for both streets to choose option C. This is where both streets are likely to have access to power. If one street chooses option C and the other chooses D, the topology will go with the defector, and the street that chose option C will be the last load to receive power if there is any remaining. This problem becomes much more interesting when you are unsure of how the other
agent in the system will act, and the game is played several times in what is known as the iterated prisoner’s dilemma (IPD).

There are several approaches to solve this game, however; no one solution is accepted as the best approach [12]. In this system, I will set up a master/slave orientation on the agents, where the bus with the higher priority is the ‘master’ and the other is considered the ‘slave’. The master agent makes its decision purely based on priority. If its priority is larger than the predetermined threshold, than it will always choose to defect. If it is below the threshold than it will decide to confess.

One approach for the slave agent would be to always follow the actions of the master agent. Alternatively, the slave could act under the rules of an IPD strategy. These include, AIID (always defect), Tit For Tat (cooperate on first move then copy opponents last move), Grim trigger (cooperates until the opponent defects, then always defects), and so on. Because the master agent makes decisions based on priority it is difficult to make the opposing agent respond with guaranteed success. Therefore a Pavlov strategy was implemented on the second agent.

2.3.2 The Pavlov Strategy

In the Pavlov strategy, the agent chooses C on the first turn. Next, if it is rewarded (scores 3 or 5) it makes the same move; otherwise it makes the opposite choice. To observe the success rate of the Pavlov method we constructed the IPD game using MATLAB software. In this game the Pavlov agent will play against a master player making a random selection. Overall, the Pavlov agent was able to maintain reasonable score during the game. On average, after 100 iterations of the game, the Pavlov agent
scored about 15 fewer points than the master agent and occasionally scored higher. The problem with this approach is that Pavlov’s method is designed to affect the behavior of the opposing player, which cannot happen if the second player makes random choices. Thus testing this method against a random player does not produce an interesting or useful result.

As mentioned before, the master agent will behave purely on based on its priority and therefore will not act as a random player. That being said, there are several ways one could approach setting the priority. However, in our system, the priority is predetermined based on the importance of the load during a given time interval. Thus it would be counter productive to change its priority to force a more interesting result.

Figure 6 shows the SOC of both battery storage systems over one thousand intervals with the Pavlov method integrated into the software simulation of the dc microgrid. The sources consist of a PV array (40W) and a battery storage system (120Ah) each connected to streets 1 and 3. To see how the system would respond to
drastic changes in generation, both PV arrays are set to turn off for the first two intervals of a five-interval cycle. So the batteries are solely responsible for providing power for two out of every five hours. All of the loads are set to their minimal values (3W for residential, 5W for commercial).

Note that the system succeeds in maintaining the charge of the batteries in the acceptable range by adjusting the switch states when the difference in SOC is large enough. Loads with lower priority are shed first, thus the battery storage system can remain in balance. Overall, this control algorithm is successful in completing the system objectives, however; the conditions to run the IPD game only occur a few times over the thousand iterations.

With such a small frequency of initiating the IPD game, there is not enough data to determine the usefulness of the Pavlov and master agents. To reduce cost and improve results in the algorithm, an AIIC (always cooperate) technique is a more appropriate choice for the slave agent. Here the slave will always match the topology preference of the master agent if there is an incompatible network selection.

2.4 Consensus Methods

In order to test the success of the aforementioned control technique, a second controller will be developed. This way we are able to get a broader sense of how well each of the controllers performs in the three system objectives. Knowing the strengths and weaknesses of each method provides critical insight on the design of a well balanced controller.
The agents in this system fall under the category of a consensus problem, in that all of the processes in the group need to agree on a specific value based on the votes of each process [13]. Consensus problems are an excellent way to approach distributed systems because are able to make group decisions using limited message paths. The conditions for a consensus problem are that all agents must agree on the same value, the value must be submitted by at least one of the agents, and lastly, there is at least one initial system state that produces each output [14]. These three conditions define the safety requirements for consensus. The second requirement for consensus is eventual liveness, which makes sure that if a value is chosen, that value is eventually learned by the system.

Our problem will address an asynchronous distributed system, where there are no limits on the execution of a process, and there are no bounds on transmission delays. Asynchronous problems are not considered solvable for consensus because failures and message delays. Conversely, synchronous distributed systems receive messages in a bounded time. The drift of each agent’s local clock has a known bound, so the timing between agents has a common protocol. Finally, processes are able to fail by stopping.

2.4.1 The Paxos Algorithm

The Paxos algorithm is a common technique used for solving consensus problems similar to the one we are faced with. This algorithm uses three classes of agents, proposers, acceptors, and learners. Note that a process in the controller has the ability to act as one or more agent. The agents exchange information by sending messages to each other. The time constraints of these massages may differ based on the synchronicity of
the system. Described below is a brief version of Leslie Lamport’s *Paxos Made Simple* synchronous non-Byzantine algorithm[15].

The first step in Paxos is known as the Election phase, where a leader is chosen. Here the proposers select a proposal number $p_i$, and send a prepare request with the proposal number to a majority of the acceptors. Then, when an acceptor receives the prepare request it accepts only the proposal with the highest request number. Also, proposers are allowed to make multiple proposals, as long as the entire algorithm is followed each time.

Phase two of the Paxos algorithm is referred to as the Bill phase. This states that if a proposer receives a response to its prepare request from more than half of the acceptors, than the proposer responds with an accept request back to those acceptors. The accept request contains both the proposal number $p_i$ and its value $v$. Next, if an acceptor collects the accept request with a proposal number $p_i$, it accepts the proposal. However, if it has already accepted a proposal with proposal number greater than $p_i$, than it will not accept the later proposal.

The third and final movement of the Paxos algorithm is the Law phase. Here the system learns which value the acceptors have chosen. To accomplish this, each acceptor responds with their choice to each of the learners. Each of the learner agents then sends a message to the other learners with the chosen value. This method requires several responses (the product of the number of acceptors and learners). One alternative would be to select one distinguished learner to whom all acceptors send their response. This method only requires a response equal to the number of acceptors. However, it adds
another step, in that the distinguished learner must send the results to all of the other learners. Typically, the Paxos algorithm avoids having the information in the hands of a single agent, in case of a fault. So although the distinguished learner method requires sending fewer responses, it is certainly less reliable.

The most practical approach to the Bill phase is to compromise the two ideas, and have the acceptors send their responses to a small set of distinguished learners, thus reducing the amount of messages being sent while still maintaining an acceptable level of reliability in the algorithm. However, it still requires an extra step in communication. Therefore depending on the need of the system one method of the Bill phase may be more useful than another.

Due to the simplicity and usefulness of the Paxos algorithm, many researchers have produced alternative versions for different system configurations and objectives. These include Boichat’s FastPaxos [17] algorithm, and Martin and Alvisi’s Fast Byzantine Paxos (FaB) [18]. Both of these methods are useful because, as their titles imply, they only require two communication steps. However, any these techniques need to be modified to implement on our lab-scale microgrid due to the amount of available components on the grid.

Boichat’s FastPaxos [16] is a method optimized to run only during stable and fault-free periods. In periods of stability, there are no process crashes or recoveries. Here, once a leader collects a proposal, it takes just two communication steps until the proposal can be accepted. Lamport also created a speedier communication method called Fast Paxos[18] (with a space in between) that optimizes the solution during stable periods.
Again, this process only requires two communication steps. The issue with both of these methods is that they require too many agents, and are overcomplicated for our system. For example, FastPaxos uses a round-based consensus, where a read and write phase are used to determine if a certain value was already proposed, and writes a new value or repeats the old value in the case that the value was previously proposed. Our Paxos algorithm will follow Lamport’s Fast Paxos method with some additional simplifications.

2.4.2 Byzantine Fault Tolerance

One useful extension to the Paxos algorithm is Byzantine fault tolerance method. Byzantine fault-tolerance, described by Lamport, Shostak, and Pease in The Byzantine Generals Problem [19], is a useful addition to the Paxos algorithm because it prevents faults and miscommunications in distributed networks. The set up is defined abstractly as a group of generals in the Byzantine army who are camped with their troops around an enemy territory. Communicating only by messenger, the generals must agree on a common battle plan. However, one or more of the generals could be traitors who try to confuse the others.

The problem here is to create an algorithm that ensures that the loyal (non-faulty) generals will reach an agreement. This solution is achieved if and only if more than two thirds of the generals are loyal. The Byzantine General’s algorithm guarantees that, all of the loyal generals decide on the same plan of action, and a small number of traitors cannot cause the loyal generals to adopt a bad plan.

The first condition is achieved by having all of the generals use the same method for collecting information. Likewise, the second condition is achieved by using a robust
method. An example of this would be, if the agents are trying to make a binary decision (1 or 0), then the decision is be based on the majority of votes from the agents. A small number of malicious agents can affect the decision only if the decision is split between the loyal agents almost evenly. In this case, neither decision can be bad because several loyal agents have voted for each decision variable.

Unfortunately, the solutions for byzantine fault-tolerance are often inefficient, and require several communication steps. One method of reducing the cost of the algorithm is to make assumptions about the type of fault that may take place in the network. For example, a computer may fail to respond to a message, but will never respond incorrectly. For systems that necessitate high reliability, assumptions cannot be made, and the full byzantine fault-tolerance method must be used.

Our dc microgrid must be able cope with the failures of one or more of its components. In a real world system, messages sent over large distances are prone to have faults and miscommunications. A real world system would become more reliable using some method of byzantine fault-tolerance. For the purposes of the lab-scale dc microgrid, there are not enough agents to implement any meaningful methods of byzantine fault-tolerance. Therefore we will leave the subject as a topic of future work.

2.4.3 FAB Paxos

According to Martin and Avisi’s Fast Byzantine Paxos (FaB), the consensus algorithm is still able to function even in the event of a number, \( f \) faults as long as there are at least \((5f+1)\) acceptor agents[17]. FaB Paxos runs similarly to the Fast Paxos algorithm in that it only uses two communication steps, however; it requires more agents
to be able to prevent failures. Specifically, if some proposal is sent \((v, p_n)\), and is received by the acceptors at least \([(a-f+1)/2]\) times in the progress certificate, then the progress certificate vouches for only \(v\) at \(p_n\). If such a pair does not exist, the progress certificate vouches for any value as long as the proposal number matches the one in the progress certificate.

FaB Paxos requires the number of acceptors to be five times greater than the number of faults \((a>5f)\), and the number of proposers to be three times the number of faults \((p>3f)\). There is an alternative method created by Martin and Alvisi called Parameterized Fab Paxos [20], which requires \(3f+2t+1\) acceptors, and can tolerate up to \(f\) number of faults. Again, these methods require too many agents to integrate in our study. Therefore a simplified version of these techniques will be implemented.

A similar extension for preventing malicious agent(s) from making the correct decisions is Lamport’s \(BPCon\) algorithm. This Byzantine Paxos algorithm or \(BPCon\), starts out the same way as the classical algorithm.

P1. The acceptors can vote for a value \(v\) in ballot \(b\) only if \(v\) is safe at \(b\).

P2. Different acceptors are not able to vote for different values in the same ballot.

Phase 1a. The ballot-\(b\) leader sends a 1a message to the acceptors.

Phase 1b. An acceptor responds to the leader’s ballot-\(b\) 1a message with a 1b message containing the number of the highest numbered ballot in which it has voted and the value it voted for in that ballot, or saying that it has not cast a vote.
Phase 2a. Using the $1b$ messages sent by a quorum of acceptors, the leader selects a value $v$ that is safe at $b$ and sends a $2a$ message containing $v$ to the acceptors.

P3a. If no acceptor in the quorum has voted in a ballot numbered less than $b$, then all values are safe at $b$.

P3b. If some acceptor in the quorum has voted, let $c$ be the highest-numbered ballot less than $b$ in which such a vote was cast. The value voted for in ballot $c$ is safe at $b$. (From P2, there can only be one such value.) [20]

In order to tolerate faults in the system, Lamport defines a set of $byzacceptors$ [20] as the union of the sets of real and fake acceptors. Also, the $byzquorum$ is defined as the set of $byzacceptors$ that is guaranteed to contain a quorum of acceptors. For a quorum of some number $q$ acceptors, the $byzquorum$ contains $(q + f)$ byzacceptors.

The most important step in the Paxos algorithm is the Phase 2a message sent by the leader. This step informs all of the acceptor agents that a value has been chosen. Steps P3a and P3b ensure that the chosen value is safe. P3a holds if it receives $1b$ messages from a $byzquorum$, each asserting that the sender has not voted because the $byzquorum$ contains a quorum of acceptors. However, P3b creates an issue because there is no way to differentiate between a real and fake acceptor. Safety can be upheld by requiring a vote to be reported in the highest-numbered ballot $c$ by $f + 1$ byzacceptors. Unfortunately, this causes the liveness property to fail because it is possible to reach a state in which this condition does not hold for the $1b$ messages to sent by the real
acceptors. Thus, the BPCon algorithm is not a perfect solution for our system set-up because we lose the liveness property.

For the purposes of this study, we will not include any fault tolerance methods. Fault tolerance techniques require too many acceptor agents to protect against malicious agents in our system. We simply do not have enough components on our hardware to implement any effective fault tolerance or byzantine system. It is however, an important aspect of the security and reliability of the grid, and therefore should be considered in the future work of this project.
Chapter 3: DC Microgrid Simulation

3.1 DCMG Paxos Simulation

For the model dc microgrid, the initial algorithm for software simulation makes some small adjustments to the original Paxos method. First, for the time being we will exclude all fault-tolerance and byzantine extensions. Our network does not have the required amount of components to meet the standards of byzantine fault-tolerance. Thus the system will be in the stable state for the entire simulation. This will provide some insight on how the dc microgrid reacts to Paxos at its most basic level.

As stated in the original decentralized algorithm, every load is designated priority for all iterations. Similarly, priority will be determined based on the loads importance to the grid during a given time interval. Next, all of the streets will send in their proposal numbers $p_n$ to a single acceptor agent. The acceptor agent receives the proposal numbers and selects the street with the highest proposal. After the election, the streets go through some network topology process as the decentralized algorithm to determine which supply source they would refer to connect to. Then they send their values to the register agent, which uses those values along with the leader selection to determine the correct network configuration. The five network configurations are:

1. All streets connected to source (1).
2. Streets 1, 2, 4 connected to source (1), and street 3 connected to source (2).
3. Streets 1, 2 connected to source (1), and streets 3, 4 connected to source (2).
4. Streets 2, 3, 4 connected to source (2) and street 1 connected to source (1).
5. All streets connected to source (2).

Although the choices for supply power are binary (source 1 or 2), there are often multiple configurations that will satisfy the decision. The register takes the value, \( v \), sent by the proposer, and determines which configurations are accepted by that value. For example, if street 4 proposes a value of \( v_4 = 2 \), the acceptable configurations are 4 and 5. Configurations 1, 2, and 3 are not accepted because they require street 4 to be connected to bus 1. This process evaluates each street, and counts the votes for each configuration. If the configuration with the most votes is accepted by the leader, then that value becomes the chosen configuration. If the leader does not accept the value, but is out-voted by a three-fourths majority, then that is still the chosen configuration.

In the event of a tie, the selection goes to the configuration that is accepted by the leader, if all or none of the choices are accepted then the agent selects the more conservative configuration (e.g. configuration 3 is more conservative than 1 because

![SOC vs. Time]

**Figure 7 - Battery SOC using the Paxos Algorithm (with constant load priorities)**
configuration 1 puts all of the load on bus 1). Although a tie in the register is relatively unusual, setting the configuration to the conservative value is important in maintaining the reliability of the system. Placing the entire system load on one source induces risk in the case of a fault or failure. After the configuration has been chosen, the simulation runs the same resource allocation process, and updates the battery storage system as the original algorithm.

Figure 7 shows the results of the Paxos algorithm. At the start of the simulation, the battery SOC on both busses behaves the same way as the original decentralized controller. The battery with more storage discharges, while the lesser battery holds less load and charges. This continues for several iterations and eventually the SOCs converge, where another configuration is made, and the system becomes more balanced. During this time the batteries diverge, but at a much slower rate. It is safe to assume that after many iterations, when ΔSOC is large enough, another configuration will be chosen and the process will begin again. Overall, the controller is successful in managing the battery SOC, and requires minimal switching. In this simulation the load was set low enough that no load shedding was necessary, however the algorithm still ensures that power is distributed in order of priority. Thus the main three system objectives are successfully met.

3.2 JADE

Now that the Matlab simulations have been created, the next step is to build the agents within a software platform designed for distributed communication. Then we will be able to run those agents with along with the Matlab simulation through TCP sockets
and confirm our results. Once the agents behave as expected we will be able to run the decentralized agent system on the dc microgrid. The software platform chosen to complete this task is JADE (Java Agent Development Framework).

JADE is a software framework completely implemented in the Java language [21]. It uses a middleware that simplifies the application of multi-agent systems that complies with the FIPA specifications [22]. FIPA (The Foundation for Intelligent Physical Agents) is the IEEE Computer Society standards organization for agent and multi-agent systems and the interoperability of its standards with other technologies. JADE provides useful execution in peer-to-peer asynchronous agent communication.

JADE sets up a runtime environment, where the agents are active on a given host. Each instance of a JADE runtime environment is called a container. This is where multiple agents are housed. The platform is defined as the set of all active containers. One container must be defined as the main container and must always be active in a platform. The main container is the first container to start on the platform, and all other containers register with it as soon as they start. In addition, the main container also holds two unique agents. The AMS (Agents Management System) makes sure that each agent in the platform has a unique name. The DF (Directory Facilitator) helps agents find the other agents who can provide the services he requires.

Agents are able to communicate with each other if they are hosted in the same container, different containers, the same platform, or even different platforms. The asynchronous message format ACL, is defined by FIPA. An ACL message is composed of a sender, receiver(s), a communicative act, and the content. The communicative act
interprets the intention of the sender of the message. Lastly, the content which is the actual information sent by the message.

3.2.1 Implementing Control Methods Within JADE

To setup the control functions in JADE we run all agents under the main container. For a system of this size, there is no added benefit in creating multiple containers. Communication between agents is achieved using Jade ACL messages. The `setContent` and `addReceiver` commands create a message of the String type and send the messages to the desired agents. In order to receive a message, a message template must be created using the `MatchPerformative` and `MatchSender` commands. This prepares the agent to receive an ACL message from a specified agent. Then the receiving agent must be told how to react when a message is sent. To do this a `WakerBehaviour` is set. The `WakerBehaviour` performs one instance of a task after a given timeout has passed [23]. Using a short timeout period ensures that all of the messages to the agent have been sent. Running the agent immediately can cause missed messages resulting in a ‘null pointer exception’. Within the behavior, messages are finally able to be received with the `receive()` method, returning the message if there is one, or null otherwise [23]. Keep in mind that ACL messages are of type string, so all of the data being passed in the agents must convert from type double to type string when sending messages, and from type string to type double when receiving them. Lastly, at the end of each behavior, a primitive `block()` is called to set a delay for the following execution of the behavior.

For the decentralized controller, we have an input agent, which sends the values of the priority load schedule, PV schedule, battery SOC, and current network
configuration to the load agents. Each bus will act as its own agent, and run its network topology process to determine which source it would like to connect to. Then all of the topology values are sent to the register agent who examines the choices and implements the master/slave operation if there is a network impossibility. The register agent, labeled as the output agent, acts as the output as it displays the final network configuration and sends the decision back to Matlab. The communication paths are shown in Figure 6 where $C_a$ is the cost to purchase power from generation source $a$, $C_b$ is the cost to purchase power from source $b$, $Pen_{1-4}$ is the penalty for switching on streets 1-4, and $PL_{1-4}$ is the updated source connection of streets 1-4. Once the network topology algorithm has completed, in Matlab the first phase of resource allocation will initialize and each street will receive the amount of power it will receive for that iteration, as well as the amount of power charge/discharged from the battery. Finally the second resource

Figure 8 - Agent Communication Flow for Network Topology Algorithm
allocation will take place, and the individual loads agents will receive a message with their allocated power consumption. After that, the network topology and resource allocation information is sent to the hardware over an OPC server.

The Paxos controller will operate slightly differently than the decentralized controller. The bus, source, and battery agents will still receive the input information from the input agent. Then, once the bus agents have received additional messages from the source and battery agents, they will send proposals to a register to determine which street will act as the leader for that iteration. The register returns the value of the leader to every bus agent in the network. Next, the non-leader busses for that iteration send their preferred source locations to the leader, who will decide which configuration pleases the majority of the streets. If there is a tie for the most valued configuration, the process is the same as before and leader will select the configuration that it would like. If the leader is for or against both choices in the tie it chooses one of the two configurations randomly because they are equally valued within the system.

3.3 Running the Java/Matlab Controller

As stated before, the communication loop begins in Matlab, with the initial conditions. These include, current network configuration, load schedule, load priority schedule, initial battery SOC, and solar PV schedule. Then, using TCP sockets, the Matlab input information is formatted into a single message and sent to the JADE input agent. The JADE input agent uses a TickerBehaviour() to listen for incoming messages every 100 milliseconds. When it receives a message, it parses the message and checks for the proper beginning and end of the message. If a complete message is received, the
input agent distributes the necessary pieces of information to the remaining storage and load agents. If an incomplete message is received, the input agent will return an error message to Matlab, which prompts Matlab to resend the message. This process continues until a complete message is sent. This technique acts as an alternate solution to having a socket buffer within JADE.

After the input agent sends the first message, the storage and load agents receive the message using a oneShotBehaviour(). This allows the agents to trigger in a cascading effect, where agents can receive information, process it, and send a new message just before the following agent(s) are triggered to do the same. After all of the messages have passed, the output agent, which acts as the server for our modified Paxos algorithm, determines the configuration, and returns the selection to the input agent. The input agent then sends the final message back to Matlab over the same socket in which it received the message.

Throughout this period, Matlab runs a socketRead() function which attempts to read any data on the socket. By setting blocking equal to true, the program will continue to execute until some data is received. Then the message is evaluated. Error messages will cause Matlab to resend the original message; otherwise the message is processed and sent through the remaining two resource allocation algorithms. Once the full iteration of the algorithm is complete, Matlab simulates the updated SOC and monitors the distributed power for each load. Then the algorithm repeats for the following iteration.

To ensure that the JADE agents have enough time to pass their messages, the oneShotBehaviour’s are spaced so that the entire process takes exactly 4.5 seconds.
Therefore, after the first iteration, a tickerBehaviour() can be used in each agent's sequence of behaviors to repeat the cascading process every 4.5 seconds. This result causes the second phase of simulations to be rather slow, but we examine that the Matlab/JADE simulation runs consistently and similarly to the first phase Matlab simulation.

3.4 Simulation Results

![SOC vs. Time](image)

**Figure 9a – Battery SOC using the Decentralized Controller (random load priority)**

**Figure 9b – Battery SOC using the Paxos Controller (random load priority)**
The results of the Matlab simulations are shown in Figure 7 a and b. Here the battery SOC vs. time is plotted for each battery storage system for 1000 iterations (one hour long iterations). The priority for each bus is a randomly chosen real number between 0 and twenty. The actual priority schedule will certainly not be selected at random; however, the random schedule allows us to test the controller performance under very volatile conditions. For both simulations, the total load was set at 40W. At this amount of load, both controllers were able to balance the storage devices SOC as well as distribute power to all loads. Thus no load shedding is necessary. Both controllers appear to respond in similar ways. The mean values of the battery SOC during the simulation are shown below in Tables 2 a and b.

<table>
<thead>
<tr>
<th>Paxos</th>
<th>Mean SOCa (%)</th>
<th>Mean SOCb (%)</th>
<th>Mean Diff (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>70.35</td>
<td>70.51</td>
<td>0.16</td>
</tr>
<tr>
<td>2.</td>
<td>70.52</td>
<td>71.33</td>
<td>0.19</td>
</tr>
<tr>
<td>3.</td>
<td>68.73</td>
<td>68.91</td>
<td>0.18</td>
</tr>
<tr>
<td>Average</td>
<td>69.87</td>
<td>70.25</td>
<td>0.18</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dec</th>
<th>Mean SOCa (%)</th>
<th>Mean SOCb (%)</th>
<th>Mean Diff (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>71.39</td>
<td>72.07</td>
<td>0.68</td>
</tr>
<tr>
<td>2.</td>
<td>73.25</td>
<td>74.62</td>
<td>1.37</td>
</tr>
<tr>
<td>3.</td>
<td>72.49</td>
<td>72.89</td>
<td>0.40</td>
</tr>
<tr>
<td>Average</td>
<td>72.38</td>
<td>73.19</td>
<td>0.82</td>
</tr>
</tbody>
</table>
In the tables above the mean value of each battery SOC as well as the mean difference between the two batteries are shown. Over three trials, the Paxos algorithm maintains a lower SOC than the Dec algorithm, but maintains the ΔSOC under a smaller mean difference. Also, the Paxos algorithm produces a greater average number of switches than the Dec algorithm with 33 and 22 switches respectively. Note that the trade off for minimizing switching is increasing the mean difference between the two batteries. If the system requires tight bounds on battery SOC then more switching will be necessary.

It appears that the Paxos method is the higher performing controller for managing the battery SOC, while the Dec method activates fewer topology switches. The Paxos method has a more dynamic decision making process, which requires more communication steps then the Dec method. The Paxos algorithm is better suited for larger systems. For example, if there are more than two generation sources then the basic decentralized algorithm will fail in the network topology algorithm. The Paxos method can be scaled for any number of network configurations, as long as a majority decision is made. Also, as the number of voting agents increases, the fault tolerance in the system will also increase. In a real world system the Paxos method is the preferred choice for network planning, however for our dc microgrid, the decentralized method will produce similar results and with less communication steps.
3.4.1 Load Shedding

An important feature of the control algorithm is to distribute power in order of priority. In an emergency islanding mode, a shortage of power may occur during a given planning interval. As a result, some load must be shed. To determine which loads will be shed first, a unique priority value is assigned to each load for every time interval. As stated before, higher priority yields greater importance to the network. For example, a hospital facility or water treatment plant will be assigned higher priorities than a residential home, because they are considered more important to system during a specified time interval.

If power supply is able to manage the load requirements for during the simulation, then no load shedding is necessary. However, if there is a slight imbalance between load and supply, load shedding will be required frequently to maintain the SOC requirements of the batteries. In order to implement this in the resource allocation algorithm, the street bids will be generated from the cost of the battery, the SOC of the battery, the priority of the street, and a gain term $g$. The gain is a linear term that calculates the minimum acceptable priority before being dropped. When the battery storage levels are 85% or greater, the sum of the gain, and the SOC will be greater than one, regardless of the priority. Therefore, the controller guarantees that no load will be shed while battery storage levels exceed 85%. Conversely, when battery storage levels drop to 35%, the gain term drops such that only loads with priority greater than 0.9 will be cleared for power distribution. Using the linear gain term allows the controller to manage the resource allocation for any load schedule. Because the cost of the battery and the battery
SOC are the same for all bids connected to the same power supply, the streets will still be ranked in order of priority. For example, a street bid to power supply $a$ is calculated as

$$B_{da} = Ca(g + (SOCa / 7) + p)$$

$$g = 1.2 \cdot SOC - 0.9125$$

where $Ca$ is the cost of battery $a$ (calculated in step 1 of the Network Topology algorithm), and $SOCa$ is the charge of battery $a$ (divided by the maximum capacity), and $p$ is the priority assigned to the street. Note that if the gain $g$ is greater than or equal to 1, then the bid will always be greater than $Ca$, and thus the load will never be shed. The gain must be set carefully in order to balance the system.

The figures below demonstrate the load shedding function of the controller. In this simulation a total load of 48 W request power for each hour-long time interval. With two 12 V solar arrays running of a schedule of twelve hours on, then twelve hours off. This schedule will roughly model the daily sunrise and set, assuming there is maximum sunlight each day for twelve hours. Again, we will start the batteries at 60% and 80% capacity. Priorities for the streets will be set randomly for each time interval. As stated before, random prioritizing increase the volatility of the system, and will validate the controller’s ability to manage the system under harsh conditions.
conditions. First, without load shedding, it is clear from Figure 8 that the system is not able to maintain the battery SOC conditions. Although all of the loads receive full power for each iteration, the battery storage devices will soon be depleted, resulting in a dangerous lack of reserve power. Without reserve power no loads will be able to receive power during the twelve-hour dark period, and the solar PVs will not be able to support the entire load during the light periods. Therefore, load shedding is a crucial part of islanded microgrid systems. Although some load will have to be dropped, load shedding will allow the system to maintain reserve power and guarantee that the most important loads in the network receive power. If overloading were to continue at the current rate in Figure 10, the storage components would oscillate around 35% charge, as the batteries charge/discharge and occasionally provide power to a few high priority loads. This situation is less than ideal, but it ensures that the storage limits are met, while doing its best to provide power to the most crucial loads. Finally, note that any load will be guaranteed to receive power if its priority is set at 1. A priority of 1 will bypass the gain term and ensure that the load receives power before the battery or any other loads. If the system requires that more than one load always receive power, than the loads need to be set at unique values greater than 1. This will prevent any potential ties if the crucial loads are competing for the same power provider. With load shedding turned on, the batteries are able to maintain their SOC, as observed in Figure 11.
Figure 12 a-d shows the resource allocation for each street in the system. For each iteration, the power sent to each street as a percentage of the power requested is plotted. For many of the iterations, the street receives 100% of the power. When the load needs to be shed, partial or no power is allocated to the low priority loads. In this simulation, the ratio of power supply to load demand requires a large amount of load shedding. Over the ten-hour simulation, the four streets receive their full power request 26 times. Thus there are 14 instances where load is shed. As the supply and demand ratio improves, less load shedding is required.
Once the streets have received power, the second round of resource allocation begins. This time each load is assigned a priority 1-3. Once again, the power is distributed to the loads in order of priority. This bidding process is much simpler this time because there is only one power supplier, and there is no battery to charge or distribute power. Rather than producing an additional twelve plots for the resource allocation of the individual loads, Table 3 shows the percentage of requested power for each iteration 1-10. For this simulation, loads receive the full amount of requested power for most iterations. Clearly no utility would be satisfied with the amount of load being shed, but keep in mind that this simulation is exploring the situation of an emergency islanding mode. In this case, there is not enough power to supply all loads and maintain
the battery SOC. Also note that due to the changing priorities, the loads that are dropped first change with every iteration. With this set of initial conditions, the system was able to provide 74.3% of the load demand over the ten-hour simulation. Obviously, decreasing the load demand, or increasing the power supply will decrease the amount of dropped load.
Table 3 - Load Shedding for Stage 2 Resource Allocation

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Load 1</th>
<th>Load 2</th>
<th>Load 3</th>
<th>Load 4</th>
<th>Load 5</th>
<th>Load 6</th>
<th>Load 7</th>
<th>Load 8</th>
<th>Load 9</th>
<th>Load 10</th>
<th>Load 11</th>
<th>Load 12</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0.05</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>0.59</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
<td>0.918</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Averag e</td>
<td>0.750</td>
<td>0.637</td>
<td>0.717</td>
<td>0.577</td>
<td>0.611</td>
<td>1.000</td>
<td>0.918</td>
<td>1.000</td>
<td>1.000</td>
<td>0.387</td>
<td>0.640</td>
<td>0.387</td>
</tr>
</tbody>
</table>
3.4.2 JADE Simulation Results

Once the JADE agents are created and tested in parallel with the Matlab code, we are able to obtain simulation results as before. Since each iteration takes 4.5 seconds, the Java/Matlab simulation takes much longer to execute. However, for hardware purposes the controller needs to run in real time, and therefore speed will not be an issue. Figure 13(a) shows the SOC of both batteries over 400 iterations. Notice that the system acts the same as in the first phase of simulation, where the SOCs converge and make the necessary topology changes to maintain balanced storage. Again, this algorithm uses the same load shedding algorithms, which produce similar results to the phase 1 Matlab simulation. Another measure we may observe in this simulation is the configuration selections for each iteration in Figure 13(b). The y-axis marks each configuration selection (1-5) for each iteration on the x-axis. This figure allows us to get an idea of how the topology changes in the system under a light load setting, and with random branch priorities.
Chapter 4: Hardware Demonstration

4.1 Communication Overview

As stated before, our Paxos controller will interact with the hardware system’s PLC through a CIP stack designed in the Java language. During each time period, the controller will receive the current system information (switch states, priority, battery SOC, etc.). Then the controller will automatically run a single iteration of the algorithm and return the updated network topology and resource allocation to the CIP. Afterwards, the controller will wait until the end of the next period and begin the process again.

Keep in mind that our controller only handles the high-level control elements in the network. In a sense, the Paxos controller acts as the Look-Ahead planner, which is found in more conventional grid management systems. To handle the low-level controls, like faults in the network, several ladder logic files will be implemented for each element in the system. Specifically, there are ladder protocols for the batteries, the solar PV, and each of the houses/commercial buildings. These programs communicate through tags defined in the program to make actual changes in the hardware.

Our hardware model of the DCMG, designed by Jordan Murray, is capable of testing several control and management simulations. All of the control signals on the grid are ran through a single PLC using IF-16 and OB-32 modules. The signals are converted to program tags in RSLogix 5000 and scaled to their correct values. All of the control signals and their locations on the grid are displayed in Figure 2. In addition, there are regulated and unregulated measurements for both current and voltage from each of
the source boards. The source boards control the power distribution from all of the system sources and allow power sharing for devices supplying power to the same load using dc/dc converters. Scenario-generator signals are ran through a separate PLC. This allows the control signals to have no direct information about the scenario-generator, and therefore be able to perform realistic contingency analysis and fault experiments.

4.2 Hardware Demonstration

For our initial experiment, we operate the grid in a “night-cycle”. In other words, we will have two batteries connected to Busses 1 and 3, which will supply the entire load. In the future a more realistic demonstration may be run using solar PV generation along side the batteries, however time constraints prevented us from completing the remaining source boards. Rather than use real batteries for our simulation, we decided to implement a virtual battery strategy, where the sources were supplied by a two-input power supply, and the updated battery SOC could be calculated bated on the size of the load and the nominal bus voltage for each side of the circuit. This was done in order to adjust the capacity of the battery to suit the size of the experiment we wanted to run.

In this test, we will operate the grid using six loads (two on Busses 1 and 3, one on Busses 2 and 4 with varying loads). The system will initially start in configuration 3 (Busses 1 and 2 connected to Source a, and Busses 3 and 4 connected to Source b). The virtual batteries are set to 4Ah on each storage device starting at 80% and 75% capacity. To force decision making in the initial test, random priorities are generated in Java for each iteration, while the load priorities are set to constants. The iteration time period is set to two minutes and ran for 15 iterations. Although the iterations are small and the
overall time operating the controller is short, the results of the test are able to confirm that the controller is successful in operating its objectives in real time on our hardware model.

Figure 15 (a-f) shows the SOC, configuration choices as well as the power allocated to busses 1-4. The first thing we notice in part a (top left), is that the SOC on both batteries begin to converge. This helps to confirm our storage-balancing objective. The overall trend over the entire simulation allocates more load to the battery with the greater SOC. Observing the configuration in part b (top right) emphasizes the successful load balancing, in that the controller only uses configurations 3, 4, and 5. This places a majority of the load on the source with the larger battery capacity, and switches between the configurations based on the changing Bus priorities. Lastly, parts c-f plot the resource allocation for each Bus. Although the sample size of this test is not large, the data appears to shed more load as the batteries SOC is depleted. Also, the randomized priorities makes it more difficult to determine when and where a load would be dropped. One interesting result in parts c-f is that when loads are dropped, they do not receive any percentage of their requested power. This may be caused by the lack of generation sources to assist the storage devices in providing power. If partial power were to be generated by the controller, the percent value tag is passed to RSLogix 5000 where the tag is multiplied by a period time duration (10 seconds in our simulation), which triggers the load to turn off when the percentage of resource allocation becomes smaller than the percentage of time in the period. For example, a resource allocation tag value of 0.8 would trigger the load to stay on for 8 seconds and then turn off for 2, in every 10-second period.
Figure 14 (a-f) Hardware Demonstration Results
In order to confirm that our estimated battery charging is relatively accurate, we are able to measure the current coming from each source as shown in Figure 15. Current coming from Source a is displayed in dark blue and Source b is in green. Using this data, we can measure the current (measured in amps) over one iteration to estimate the capacity drop in the battery. This technique may not be a completely accurate representation of the battery charge/discharge. Factors like Peukert’s Law and changes in temperature will also affect the performance of the battery. In a more sophisticated system, taking a measurement of the actual battery capacity at every iteration would yield greater accuracy in the controller.

![Figure 15 - Current Measured from Sources a and b](image)

Although many tests remain to explore the full scope of the controller, this initial demonstration confirms the core abilities of the controller’s performance. Future tests will run longer tests, with generation sources such as solar PV, wind, or even from an AC microgrid using a grid tie inverter. Adding more low-level control capabilities would allow for the testing of several contingency analysis and fault remediation scenarios using the labs scenario generator signals. Another component to improve the usefulness of the
controller is to complete the controller GUI. Here, a user is able to easily adjust the system settings and test the control performance of several scenarios for simulation and demonstration purposes. A beta version of the GUI is displayed in Figure 16.

![Figure 16 - Beta Version of the DCMG GUI](image-url)
Chapter 5: Conclusions and Future Work

5.1 Conclusions

The purpose of this research was to explore the planning capabilities of our labs dc microgrid. To remain in the scope of our time constraints, we studied the high level decentralized control capabilities of our system in an emergency islanding mode using two localized generation sources and two battery storage systems. The system was tested first in software simulation, and then translated onto our labs hardware model. In order to create a truly decentralized system, the Java’s agent based software, JADE, was used to build the communication framework between load, source, and storage agents. Using this technique, individual network elements are able to make important control decisions with limited information.

We have found this study to be an intriguing first look at one approach towards managing an islanded dc microgrid. As the both localized dc generation and power storage technology improves, integrating these devices on a mass scale becomes a puzzling feat. Transitioning our electrical power infrastructure from centralized ac systems to decentralized microgrids in both ac and dc will not only allow for more localized renewable energy sources, but also provide increased security. Although this study only examines one specific operating mode, (emergency islanded dc microgrid) it may act as a framework for power management in self-sufficient microgrid systems. Eventually, self-sustaining systems may become vital for bringing power to remote locations such as third-world countries or off-world colonies.
In review, the Paxos controller is comparable to the original decentralized controller for our labs system. However, for larger systems the Paxos method will prove to be more effective in making the most efficient decisions as well as having the ability to add fault tolerance extensions. The resource allocation algorithm remains the same in both controllers, and has proven to effectively shed load in order of load priority. Overall, the controller is successful in managing the system objectives, maintaining battery storage SOC, shedding load in order of priority, and minimizing switching. Although there are many simplifications in our dc microgrid model, it provides a good starting point towards more realistic and complicated networks.

5.2 Future Work

Clearly, much work is needed in order to implement this control technique on a real power system. Firstly, a great deal of testing may be done on our lab’s microgrid. To make the simulation more realistic, line loads should be added in between sources, loads, etc. This would ultimately effect the decision making process of the controller, by analyzing the potential losses for each configuration of the network, and choosing a route that minimizes losses, while still managing the three other system objectives. In addition, several contingency analysis studies could be done to test the resiliency of the microgrid. These may include line faults, generation outages, faulty switches/sectionalizers, and others. Finally, there are many other tests that may be performed on this controller, such as a changing load schedule with variable priority changes. These tests would help understand the strengths and weaknesses of the controller, and provide additional information on improving the control methodology for the next stage of testing.
Many models for microgrid systems include access to an outside generation source or a tie to another grid. This allows the microgrid to operate in both connected and islanded mode, which improves the security of the network. To enable this, a grid tie inverter may be used to connect out lab’s ac and dc microgrids. From there, many more control features would need to be added and tested. Several studies could be produced on the benefits and shortcomings of a multi-grid system.

An additional feature that could be added to the control protocol is a network re-router. This process will monitor the grid for faulty relays and line outages and reconfigure the topology to maintain the original load configuration but avoid the system fault. Luckily, because of the mesh configuration of the grid many configurations can successfully be re-routed in the event of a single fault. A Matlab program was developed to determine the new configuration switch states based on the location of a fault. In order to detect these faults on the DCMG current sensors need to be placed throughout the grid to check for unexpected outages in current. The scenario generator could then create a fault unknown to the control signals. Faults in the grid can occur at any time, and therefore our high level controller is unlikely to detect the fault within a reasonable amount of time. Thus the signal re-router program is better suited for the DCMG’s lower level controller. Due to time constrains the Matlab program was not transferred in to the ladder logic for this study and remains a topic of future interest.

Byzantine fault tolerance is additional topic of study that remains for future work. In order to secure the grid against faulty components in the system, a Fast Byzantine fault tolerance extension could be added to the Paxos algorithm to out rule \( f \) number of faulty agents assuming there are \( (5f+1) \) agents in the system. While this would improve
security in the system, it is better suited for larger networks, and therefore was not implemented in our controller.
Bibliography


