MONTHLY PROGRESS REPORT

Contractor Name:
University of Notre Dame (Michael Lemmon)

Contractor Address:
Office of Research, 940 Grace Hall, Notre Dame, IN 46556

Contract/Purchase Order No.	Task Order No.
W9132T-10-C-0008 (prime contract no.)	

Project Title:
Design and Simulation of Intelligent Control Architecture for Military Microgrids

Period Covered:
May 1 2011 – June 1, 2011

POC/COR (Reference Paragraph 5 of the SOW):

Achievements (Describe by task. Add additional tasks, if needed.):
task numbers refer to tasks in Odyssian's original contract

Task II: Model and Simulate Intelligent Microgrid

Integrated e-board load-shedding logic on simPower model of UWM microgrid.
Integrated dispatch logic on simPower model of UWM microgrid.

Task III: Distributed Control Algorithm Development

No activity

Task VI: Develop Wireless Communication

No activity

Task VII: Develop Wireless Distributed Control

Development of high-level distributed dispatcher for real power. Development of adaptive load shedding logic for Odyssian e-board.

Problems Encountered (Describe by task. Add additional tasks, if needed):

Task II: None
Task III: None
Task VI: None
Task VII: None

Open Items (List items that require action by the Contractor or the Government):s
No open items

Summary Assessment and Forecast (Provide an overall assessment of the work and a forecast of contract completion):

This month saw activity in the development of a distributed dispatch logic and load shedding logic, and the simulation of these logics on a simPower model of the UWM microgrid.

Load Shedding Logic: The load shedding logic that was originally developed for Odyssian's single-phase microgrid testbed was modified for integration into a simulation of the 3-phase microgrid testbed at UW-Madison. The main modification over the earlier logic developed for Odyssian was the introduction of an adaptive method for recomputing the frequency at which a shed load would be reconnected. This method involves having the load broadcast its status to everyone else when it disconnects from the microgrid. All loads then keep track of how much power has been shed by these loads and uses that to compute a reconnection frequency through the formula

$$f_{connect} = \min\left(f_{in} + \frac{load_shed}{2}, 60.5 \right)$$

where *load_shed* is the total amount of load that has been shed, f_{in}, is the line frequency (Hz) about 0.25 seconds after the load was shed and $f_{connect}$ is the frequency at which the shed load will reconnect to the microgrid.

This logic was implemented in a simPower component modeling Odyssian's e-board. The following pseudo-code represents the update logic used by this component.

```
Function Update(block)
    %block.Dwork is internal state of program (stored variables)
    %block.InputPort   is the input information to the program
    %block.OutputPort  is the output information generated by the program.

    %fetch stored program states
    %  output_state: what was last output by program
    %     connected (1) or unconnected (0)
    %  latch_state:  a shed load remains unconnected
    %     (i.e., latched) until it is unlateched.
    %      once unlatched the load is free to reconnect
    %      latched (1) / unlatched (0)
    %  timer_state: value of one-shot timer
    %      positive integer values
    %  freq_connect: frequency at which reconnection occurs
    %      this is a positive real-valued number

    output_state     = block.Dwork(1).Data(1);
    latch_state      = block.Dwork(1).Data(2);
    timer_state      = block.Dwork(1).Data(3);
    freq_connect     = block.Dwork(1).Data(4);

    %fetch inputs to program
    %  reset_input:  command signal that reinitializes component to its home state
    %  freq_input:    line frequency at e-board (Hz - postive real)
    %  current_input: RMS current flowing into e-board (A)
    %  voltage_input: RMS voltage across e-board terminals (V)
    %  load_priority: characterizes priority level of eboard
    %       (1) = critical  ,  (2) = non-critical
    %  shed_load:  the total amount of load (pu) that has been shed so far
```

```
    %      this is computed using the output_states of all boards

    reset_input   = block.InputPort(1).Data;
    freq_input    = block.InputPort(2).Data;
    current_input = block.InputPort(3).Data;
    voltage_input = block.InputPort(4).Data;
    load_priority = block.InputPort(5).Data;
    shed_load     = block.InputPort(6).Data;

    %load shedding update logic
    %   if the timer_state is positive, then the one-shot timer is running.
    %      So decrement the timer (if timer_state > 0)
    %      If timer_state == 0, then timer has expired (or is not running)
    %           in which case we set the unlatch the board (latch_state=0)
    %           and compute a new connection frequency (freq_connect)

    if (timer_state > 0)
      timer_state = timer_state-1;
      if (timer_state == 0)
          latch_state = 0;
          freq_connect = min(freq_input + shed_load/2 , 60.5);
      end;
    end;

    %If a reset pulse is received, then restore the home state of the board
   if (reset_input==1);
        timer_state=0;
        latch_state=0;
        output_state=1;
        freq_connect = 60.5;
    end;

    %start the timer if the output is latched
    %and frequency is greater than 60.5


    %reconnect if the output is not latched and
    %the frequencyis greater than the connect frequency.

    % if the board is not latched and the line frequency is greater than
    %     the previously computed connection frequency, reconnect the board
    if (latch_state == 0)&&(freq_input >= freq_connect)
        output_state = 1;
    end;

    %this is the load shedding logic
    %     switch on the type of load
    %     if load_priority = 1 (critical), then shed at 58.5 Hz
    %     if load_priority = 2 (non-critical), then shed at 59.5 Hz
    %
    %     when the load is shed, start the timer and set reconnect frequency
    %      to 60.5 Hz.  This connection frequency gets recomputed after 0.25 seconds
    %      (timer expiration) using the total amount of shed load and the line
    %      frequency when the timer expires.  Upon shedding the load, the disconnected
    %       status of the board is “latched”  (i.e., latch_state=1) and the output state
    %      is set to 0 (unconnected).

    switch load_priority
        case {1}
            if(latch_state==0)&&(freq_input<=58.5)&&(output_state==1)
                latch_state=1;
                output_state=0;
                timer_state = 25;
                freq_connect = 60.5;
            end;

        case {2}
            if(latch_state==0)&&(freq_input<=59.5)&&(output_state==1)
                latch_state=1;
```

```
                output_state=0;
                timer_state = 25;
                freq_connect = 60.5;
            end;
    end;

    %save the past states for the next iteration

    block.Dwork(1).Data(1)=output_state;
    block.Dwork(1).Data(2)=latch_state;
    block.Dwork(1).Data(3)=timer_state;
    block.Dwork(1).Data(4)=freq_connect;
%endfunction

function Output(block)

  output_state = block.Dwork(1).Data(1);
  block.OutputPort(1).Data = output_state;


%endfunction
```

Dispatch Logic: The dispatcher recomputes the desired P_{req} input to the UWM controllers, to minimize the cost of operating the microgrid subject to generator power limits and power line flow constraints. This update approach makes use of the augmented Lagrangian method that was tested earlier for the event-triggered dispatcher. The algorithm is developed as follows.

Define a cost function, $C(P_{gen})$, which represents the aggregate cost of running all generators. For this work, we assume C is a quadratic function of the generator power P_{gen}.

$$C\left(P_{gen}\right) = \frac{1}{2}\sum_i c_i P_{gen(i)}^2$$

We'd like to minimize this cost subject to the following constraints

$$P_{genL(i)} \le P_{gen(i)} \le P_{genU(i)}$$

$$-P_{lineB(j)} \le P_{line(j)} \le P_{lineB(j)}$$

$$\sum_i P_{req(i)} = \sum_k P_{load(k)}$$

where $P_{gen(i)}$ is the power flowing from the generator I, $P_{line(j)}$ is the power flowing through feeder line j, $P_{req(i)}$ is the power level set point for generator i, and $P_{load(k)}$ is the power flowing into load k. The first constraint requires the generator power to lie between a lower limit, $P_{genL(i)}$, and upper limit $P_{genU(i)}$. The second constraint requires the absolute value of the line power to be less than $P_{lineB(j)}$. The last constraint is a power balance condition requiring the total generators' power setpoint equal the total load power, P_{load}.

We solve this optimization problem using an augmented Lagrangian method. This involves minimizing an augmented cost function of the form,

$$L(P_{gen}, P_{line}, P_{load}) = \sum_i c_i P_{gen(i)}^2 + \sum_i \chi_i(P_{gen(i)}) + \sum_j \psi_j\left(P_{line(j)}\right) + \xi\left(P_{req}, P_{load}\right)$$

The functions, $\chi(\,)$, $\psi(\,)$, and $\xi(\,)$ in the above are terms penalizing the violation of the constraints given in the original optimization problem. The first two functions, in

particular, take the form,

$$\chi\left(P_{gen(i)}\right) = \frac{1}{w}\left(\min\left(0, P_{gen(i)} - P_{genL(i)}\right)\right)^2 + \frac{1}{w}\left(\max\left(0, P_{gen(i)} - P_{genU(i)}\right)\right)^2$$

$$\psi\left(P_{line(j)}\right) = \frac{1}{w}\left(\min\left(0, P_{line(j)} + P_{lineB(j)}\right)\right)^2 + \frac{1}{w}\left(\max\left(0, P_{line(j)} - P_{lineB(j)}\right)\right)^2$$

The last discounting function, $\xi(P_{req}, P_{load})$ will be defined in a somewhat different manner. In general, it will not be feasible to assume that generators have easy access to measurements of load power. Since the generators all use the CERTS droop controllers, one way of directly measuring the mismatch between P_{req} and P_{load} is to use the frequency droop from 60 Hz. So, rather the actual ξ function to be used will take the following form,

$$\xi(f_{gen(i)}) = \frac{1}{2\pi}\left(f_{gen(i)} - 60\right)^2$$

The dispatch algorithm searches for a local minimum of the augmented Lagrangian, *L*, using a simple gradient descent procedure. The challenge is to restructure the computation of this gradient so the generators can do it in a distributed manner. In looking at the above expression for *L*, it should be apparent that the terms involving the cost function *C* and the constraint function, χ, are easily distributed between the generators. The power balance constraint in ξ is also distributed since we compute this as a function of the frequency, $f_{gen(i)}$, computed by the generator. The only term in question is the line power constraint, ψ, since this is a function of the feeder lines rather than the generator.

To handle this problem, let us define a "transmission line" state,

$$\mu_j(t) = \frac{1}{w}\left(\min\left(0, P_{line(j)} + P_{lineB(j)}\right)\right) + \frac{1}{w}\left(\max\left(0, P_{line(j)} - P_{lineB(j)}\right)\right)$$

and let' define a generator state

$$\varphi_i(t) = c_i P_{gen(i)} + \frac{1}{\pi}\left(f_{gen(i)} - 60\right) + \frac{1}{w}\left(\min\left(0, P_{gen(i)} - P_{genL(i)}\right)\right) + \frac{1}{w}\left(\max\left(0, P_{gen(i)} - P_{genU(i)}\right)\right)$$

We've represented these states as functions of time, since they are computed from either line or generator voltages that are measured in real-time.

Following the analysis in the ACC2010 paper, we can now compute an update for the *i*th generator's $P_{req(i)}$ that takes the form,

$$P_{req(i)} = P_{gen(i)} - \frac{\gamma z_i(t)}{\pi}$$

where

$$z_i(t) = \sum_{j \in L(i)} \mu_j(t) A_{ji} + \sum_{k \in N(i)} \varphi_k(t) B_{ki}$$

where *A* is the weighted incidence matrix of the microgrid's graph defined as *A=DI.* In this equation *I* is the incidence matrix of the graph (map from nodes to links) and *D* is a diagonal matrix whose diagonal components are the reactances of the transmission lines. The matrix *B* is a weighted Laplacian matrix for the graph whose *ij*th component is

$$B_{ij} = \begin{cases} \sum_{j \in N(i)} \frac{1}{x_{ij}} & \text{if } i = j \\ -\frac{1}{x_{ij}} & \text{if (i,j) is an edge} \\ 0 & \textit{otherwise} \end{cases}$$

The preceding discussion characterizes how the dispatcher logic updates a generator's power set point, P_{req}. We now turn to discuss when the dispatcher should be adjusting this set point. If one attempts to adjust the set point immediately after an islanding event, load shedding event, or loss of generator event, then we'll see significant interference between the dispatch logic and the UWM CERTS controllers. What this means is that operating the dispatch logic during such events can reduce power quality by increasing the amount of time it takes for the CERTS controller to stabilize.

One way around this problem is to simply turn off the dispatcher during such transients. This is a realistic thing to do, since the dispatcher's role is only to optimize microgrid operation over the long-term. The dispatch logic, therefore, monitors the generator power and frequency command to detect if an event has occurred. In particular, if the line frequency drops below 59.5 Hz or if the generator power changes by more than 0.2 pu over a 0.01 second interval, the dispatcher assumes an event has occurred and it switches itself off for a specified interval of time of 0.5 seconds. This time interval gives the CERTS controller to stabilize in a way that subsequent operation of the dispatcher will not adversely impact the controller's performance.

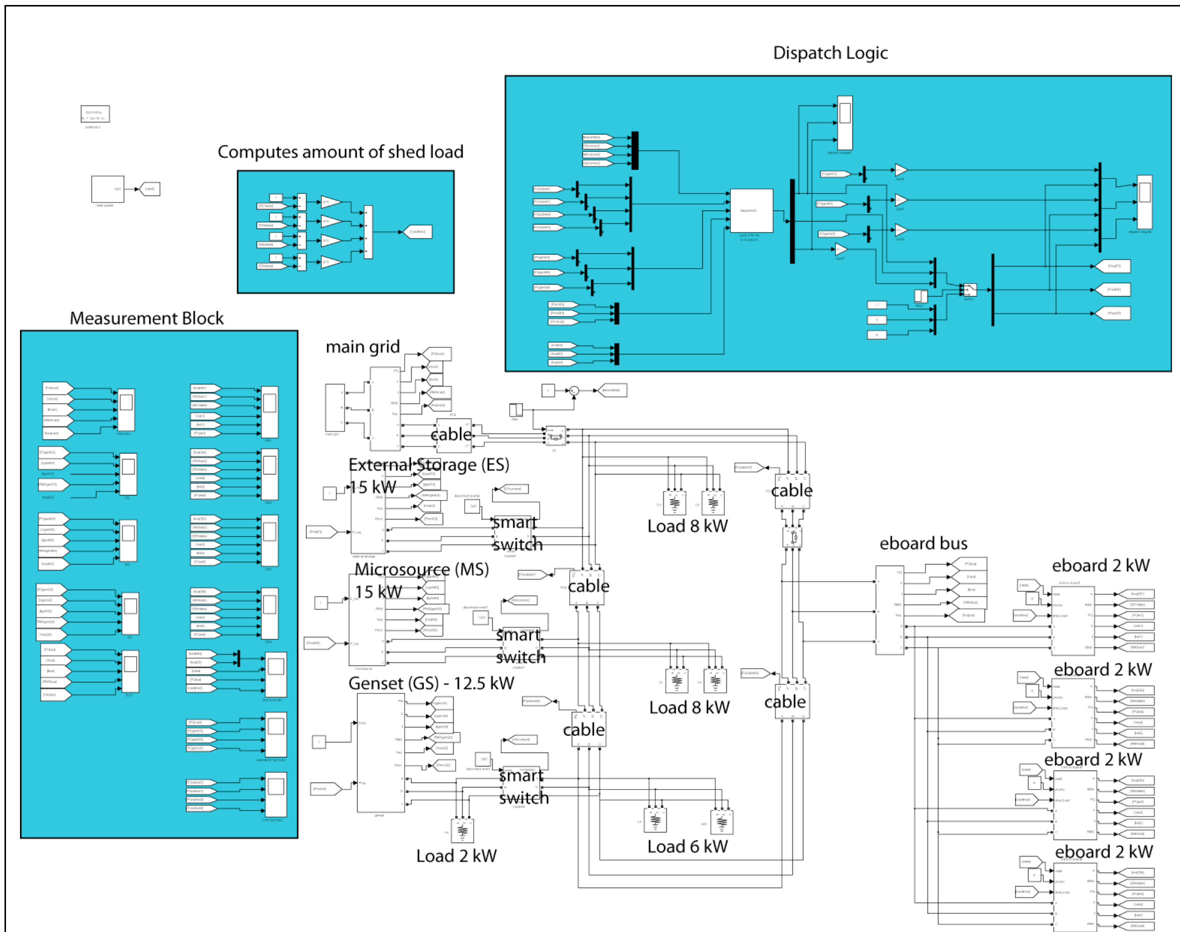
The dispatcher logic was implemented as a simPower S-function. The code for this is shown below.

```
function Update(block)
    %block.Dwork is internal state of program (stored variables)
    %block.InputPort   is the input information to the program
    %block.OutputPort  is the output information generated by the program.

    %parameters
    %   Cost(i)       = coefficients for cost of running generator i (ND)
    %   PgenUlimit(i) = upper limit on generator i power (pu)
    %   PgenLlimit(i) = lower limit on generator i  power (pu)
    %   PlineLimit(j) = limit on power flowing in line j (pu)
    %    w            = coefficient used to fix solution's tolerance
    %   gam           = update'
    %
    %   Sbase = base power of 15 kW  (used to convert measured power to pu)
    %    w0   = desired line frequency in rad/sec
    %    A    = Weighted Incidence matrix for microgrid's graph
    %    B    = Weighted Laplacian matrix for microgrid's graph

    Cost       = [1 0 0;0 .5 0;0 0  1];
    PgenUlimit = [1;1;.8333];
    PgenLlimit = [-.1; 0 ; 0];
    PlineLimit = [.5;.5;.5;.5];
    w = .05;
    gam  = 20;
```

```
    Sbase = 15000;
    w0=2*pi*60;
    Lline21 = .003300/w0;
    Lline31 = .000640/w0;
    Lline42 = .006600/w0;
    Lline43 = .004070/w0;

    A21 = 1.0/(Lline21*w0*Sbase);
    A31 = 1.0/(Lline31*w0*Sbase);
    A42 = 1.0/(Lline42*w0*Sbase);
    A43 = 1.0/(Lline43*w0*Sbase);

    A = [A21 -A21 0 0;
         A31  0  -A31 0;
         0    A42  0 -A42;
         0   0    A43  -A43];


    B = [ A21+A31   -A21    -A31      0;
          -A21    A21+A42    0    -A43;
          -A31      0     A31+A43  -A42
            0      -A42     -A43   A42+A43];


    %interal program variables
    %  Preq(i)  = last computed power setpoint for generator (i)  (pu) (Dwork(1))
    %  PgenPast(i)  = last measured generator power  (pu)  (Dwork(2))
    %  timer(i) =  timer for generator (i) (postive integer)  (Dwork(3))

    Preq              = block.Dwork(1).Data;
    PgenPast          = block.Dwork(2).Data;
    timer             = block.Dwork(3).Data;



    %external inputs to program
    %  connectState(i) = connection status of generator I   InputPort(1)
    %      connected (1) / unconnected (0)
    %  Pline(j)        = power through line j (pu)               InputPort(2)
    %  Pgen(i)         = power from generator (i) (pu)           InputPort(3)
    %  freq(i)         = commanded frequency of generator (i) (hz) InputPort(5)

    connectState(1)   = block.InputPort(1).Data(2);
    connectState(2)   = block.InputPort(1).Data(3);
    connectState(3)   = block.InputPort(1).Data(4);

     Pline      = block.InputPort(2).Data./Sbase;
     Pgen      = block.InputPort(3).Data./Sbase;
     freq = block.InputPort(5).Data;

    %temporary variables (link and node states)
    %  mu(j)  = link state for link j
    %  phi(i) = generator i's state
    %  z(i)   = combined state for generator I used in updating Preq


     %link state
     mu = max(0,(1/w)* (Pline-PlineLimit))+ min(0,(1/w)*(Pline+PlineLimit));

     %node state
     phi = Cost*Pgen + (1/pi)*(freq-60)+ max(0,(1/w)*(Pgen-
PgenUlimit))+min(0,(1/w)*(Pgen-PgenLlimit)) ;

     %z state computation

     z(1,1) = A(1,1)*mu(1)+A(2,1)*mu(2) + B(2,1)*phi(2)+B(3,1)*phi(3)+B(1,1)*phi(1);
     z(2,1) = A(1,2)*mu(1)+A(3,2)*mu(3) + B(1,2)*phi(1)+B(4,2)*phi(3)+B(2,2)*phi(2);
     z(3,1) = A(2,3)*mu(2)+A(4,3)*mu(4) + B(1,3)*phi(1)+B(4,3)*phi(3)+B(3,3)*phi(3);
```

```
    % change between Preq and Pgen
    Pdelta = gam.*z./pi;
    Pdelta = sign(Pdelta).*min(.5,abs(Pdelta)); %was originally 0.2 pu


    %switching logic to determine when to disable the dispatcher
    for i=1:1:3;

        %service one-shot timer
        if (timer(i)>0)
            timer(i)=timer(i)-1;
        end;

        %disable dispatcher if
        %    commanded frequency (freq) < 59.5 Hz (indicate loads will be shed)
        %    or
        %     the outgoingpower has changed by more than 0.2 pu in 0.01 seconds
        %   If these conditions are satisfied then start one-shot timer for 0.5 sec.
        if (freq(i) <= 59.5)||(abs(Pgen(i)-PgenPast(i))>=.2)
            timer(i)=50;
        end;

        %if the timer has expired (or is not running.  This occurs when timer=0)
        %      then you can go ahead and compute Preq (i.e. the dispatcher is active)
        %      IF the generator is connected,
        %           then use Pdelta to update Preq
        %      IF  the generator is not connected,
        %           then set Preq = 0.8 (keeps it running with a large enough frequency
        %           so the smartswitch reconnects quickly.
        %
        % NOTE: if the timer is running, then Preq is not updated.
        %       so the dispatcher is inactive and the past Preq is used.

        if (timer(i)==0);
            if(connectState(i)==1);
                %Preq(i) = Pgen(i) - Pdelta(i) - PmmLimit(i);
                Preq(i) = Pgen(i) - Pdelta(i);
                Preq(i) = max(PgenLlimit(i),Preq(i));
                Preq(i) = min(PgenUlimit(i),Preq(i));
            else;
                Preq(i)=.8;
            end;

        end;
    end;

    %save past values

    block.Dwork(1).Data = Preq;
    block.Dwork(2).Data = Pgen;
    block.Dwork(3).Data = timer;
%endfunction

function Output(block)
  %simply output the stored Preq
  block.OutputPort(1).Data = block.Dwork(1).Data;

%endfunction
```

Simulation Results: The preceding load-shedding and dispatch logic was implemented in the simPower simulation that Notre Dame has built for the UWM microgrid testbed. This section discusses some of the results in that simulation. A detailed picture of the simPower chart is shown below.



The following simulations examine the following scenario

1) Simulation starts connected to main grid (0 sec).
2) Microgrid islanding event (1 sec).
3) Loss of microsource generator (3 sec).
4) Microsource released for reconnection (3.5 sec).

Four cases were considered with these simulation: case

1) no dispatcher with reconnection of shed loads at 60 Hz,
2) dispatcher starts at 0.1 seconds with reconnection of shed loads at 60 Hz.
3) no dispatcher with adaptive reconnection of shed loads
4) dispatcher starts at 0.1 seconds with adaptive load reconnection.

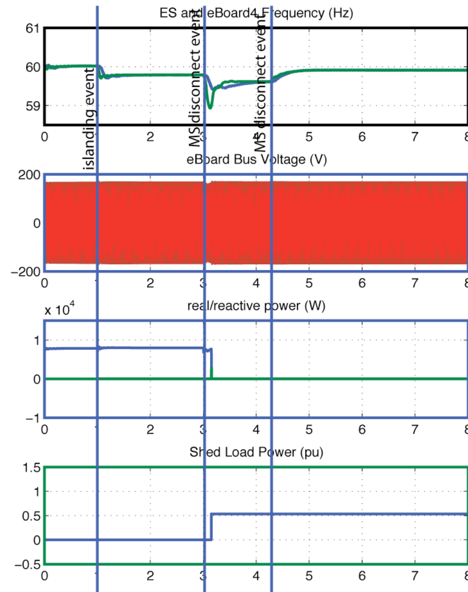
In all cases, we plot an 8 second time history of the

1) ES command frequency and line frequency estimated at e-board 4 (Hz).
2) The instantaneous voltages of all three phases at the e-board bus (V).
3) The real (yellow) and reactive (blue) power flowing into the e-board bus (W).
4) and the amount of shed load power (pu).

For those cases where the dispatcher was enabled, we also plot 8 second time histories of P_{gen} and P_{req} for the ES, MS, and GS sources.

Case 1: *no dispatcher with reconnection of shed loads at 60 Hz*

In this simulation case, P_{req} , is initially set to -0.1 pu for the ES, 0.5 pu for the MS, and .8 pu for the GS. These values are held constant through the simulation since the dispatcher is not used.

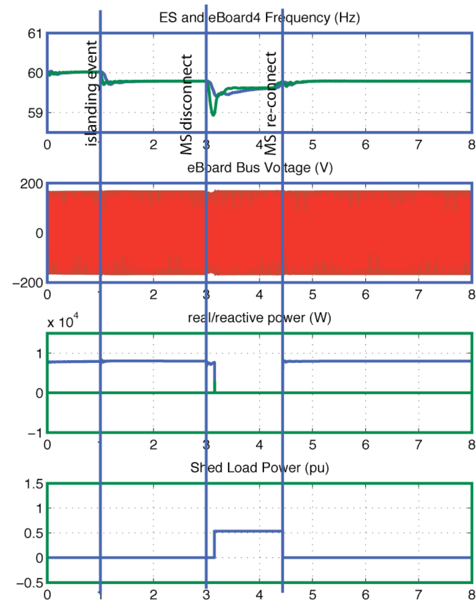


In this simulation, the e-board's shed their loads if the line frequency at the e-board's terminal drops below 59.5 Hz. The loads automatically reconnect if the estimated line frequency exceeds 60 Hz.

In this case, the eboard loads are dropped when the estimated line frequency drops below 59.5 Hz. After the MS reconnects to the microgrid, however, the line frequency never rises about 60 Hz, so the shed loads never reconnect.

Case 2: *no dispatcher with adaptive reconnection of shed loads*

In this simulation case, P_{req} , is initially set to -0.1 pu for the ES, 0.5 pu for the MS, and .8 pu for the GS. These values are held constant through the simulation since the dispatcher is not used.

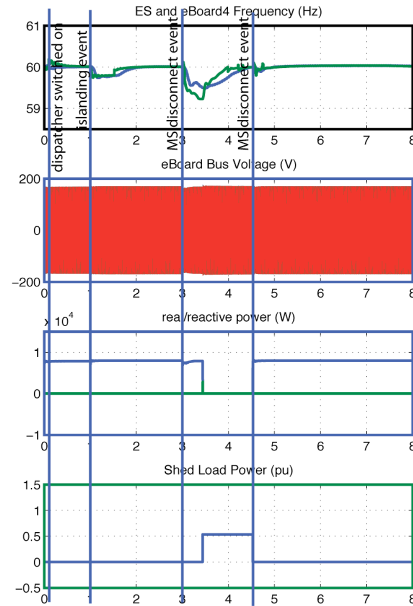


In this simulation, the e-board's shed their loads if the line frequency at the e-board's terminal drops below 59.5 Hz. The loads automatically reconnect at a frequency that is determined by the total amount of load that has already been shed.

In this case, the load reconnects almost as soon as the MS reconnects to the microgrid.

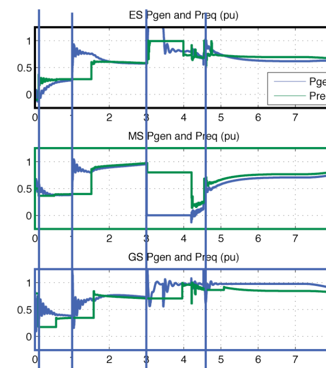
Case 3: dispatcher with reconnection of shed loads at 60 Hz

In this simulation case, P_{req} , is initially set to -0.1 pu for the ES, 0.5 pu for the MS, and .8 pu for the GS. At 0.1 seconds, the dispatcher is turned on and the dispatcher adaptively manages *Preq* throughout the rest of the simulation.



In this simulation, the e-board's shed their loads if the line frequency at the e-board's terminal drops below 59.5 Hz. The loads automatically reconnect if the estimated line frequency exceeds 60 Hz.

In this case the e-board loads are shed when the estimated line frequency drops below 59.5 Hz. The loads reconnect when the line frequency exceeds 60 Hz. This differs from case 1, because the dispatcher returns the system to its 60 Hz frequency. This can actually be seen in the frequency traces after the islanding event.



The second figure shows P_{gen} and P_{req} for all three sources. We see the generated power trying to track the commanded (requested) power computed by the dispatcher.

Case 4: dispatcher with adaptive reconnection of shed loads

In this simulation case, P_{req} , is initially set to -0.1 pu for the ES, 0.5 pu for the MS, and .8 pu for the GS. At 0.1 seconds, the dispatcher is turned on and the dispatcher adaptively manages *Preq* throughout the rest of the simulation.

In this simulation, the e-board's shed their loads if the line frequency at the e-board's terminal drops below 59.5 Hz. The loads automatically reconnect at a frequency that is determined by the total amount of load that has already been shed.

The performance of these simulations is comparable to that in case 3.