Pacific Northwest
NATIONAL LABORATORY

PNNL-37057

Experimental Study for Validation of Resilient Response of Controls

Thrust 2 Controls Assessment

November 2024

Laurentiu Marinovici, PNNL
Jan Westman, PNNL
Jeff Doty, PNNL
Celina Wilkerson, PNNL
Brett Ross, PNNL
Hannah DeSmet, PNNL
Thomas Edgar, PNNL

Prepared for the U.S. Department of Energy
Under contract DE-AC05-76RL01830

U.S. DEPARTMENT OF ENERGY

DISCLAIMER

PACIFIC NORTHWEST NATIONAL LABORATORY
operated by
BATTELLE
for the
UNITED STATES DEPARTMENT OF ENERGY
under Contract DE-AC05-76RL01830

Printed in the United States of America

Available to DOE and DOE contractors from
the Office of Scientific and Technical Information,
P.O. Box 62, Oak Ridge, TN 37831-0062
www.osti.gov
ph: (865) 576-8401
fox: (865) 576-5728
email: reports@osti.gov

Available to the public from the National Technical Information Service
5301 Shawnee Rd., Alexandria, VA 22312
ph: (800) 553-NTIS (6847)
or (703) 605-6000
email: info@ntis.gov
Online ordering: http://www.ntis.gov

PNNL-37057

Experimental Study for Validation of Resilient Response of Controls

Thrust 2 Controls Assessment

November 2024

Laurentiu Marinovici, PNNL
Jan Westman, PNNL
Jeff Doty, PNNL
Celina Wilkerson, PNNL
Brett Ross, PNNL
Hannah DeSmet, PNNL
Thomas Edgar, PNNL

Prepared for
the U.S. Department of Energy
Under Contract DE-AC05-76RL01830
Pacific Northwest National Laboratory
Richland, Washington 99354

Executive Summary

The Experimental Study for Validation of Resilient Response of Controls is a project within the RD2C Initiative. This project's objective is to execute experiments and generate datasets that are then used to quantify and characterize the resilience of different control schemes on simulated systems. This project is a multi-year effort, with the overall objective of generating datasets for each sensor developed in Thrust 2 with simulated environments and methods developed in Thrust 1.

During the second year, Opal-RT HYPERSIM was again selected as the power system simulation environment to build upon the IEEE 123-node distribution system that was modeled in the first year. Moreover, an RD2C-developed advanced control, specifically SLAC3R Cooperative Control, was integrated within this environment. Through an intense process of scenario development and testing, the model and sensors have been simulated. Additional major outcomes resulted as by-products of the scenario development and testing efforts including:

1. Improvements to HYPERSIM inverter and load models leading to more stable and realistic model behavior,
2. Development of a Python tool to automate execution of scenario tests and improve documentation of test variables using configuration files,
3. Development of a pseudo-real-time Python implementation of the SLAC3R Cooperative Control and deployment on Cybernet VMs, which improved the TRL of this strategy,
4. Detailed investigation and documentation of HYPERSIM performance issues and findings that can help reduce negative impacts to follow-on projects.
   This report documents findings, lessons learnt, and an assessment of how SLAC3R Cooperative Control impacted the resilience of the IEEE 123-node distribution system under investigation.

Acknowledgement

This research was supported by the Resilience through Data-driven Intelligently-Designed Control Inititiative (RD2C), under the Laboratory Directed Research and Development (LDRD) Program at Pacific Northwest National Laboratory (PNNL). PNNL is a multi-program national laboratory operated for the U.S. Department of Energy (DOE) by Battelle Memorial Institute under Contract No. DE-AC05-76RL01830.

Acronyms and Abbreviations

ALERT	Adaptive Learning-Enabled Resilient Tuning
API	Application Programming Interface
BESS	Battery Energy Storage System
DER	Distributed Energy Resource
DG	Diesel Generator
DSP	Digital Signal Processor
GFL	Grid Following
GFM	Grid Forming
HAGEN	Hybrid Attack Graph Generator
HELICS	Hierarchical Engine for Large-scale Infrastructure Co-Simulation
HiL	Hardware-in-the-Loop
IBR	Inverter Based Resource
IDE	Integrated Development Environment
IP	Internet Protocol
IT	Information Technology
JSON	JavaScript Object Notation
MITM	Man-in-the-Middle
OT	Operational Technology
PCC	Point of Common Coupling
PLC	Programmable Logic Controller
PNNL	Pacific Northwest National Laboratory
POI	Point of Interconnection
PV	Photo Voltaic
RD2C	Resilience Through Data-Driven, Intelligently Designed Control
SLAC3R	Self-Aware Local Autonomous and Semi-Cooperative Control for Cross Layered Resilience
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VM	Virtual Machine

Contents

Figures

Tables

1.0 Introduction

The focus of this project is to design, execute, and analyze experiments around resilience response. As part of the Resilience through Data-Driven, Intelligently Designed Control (RD2C) initiative, this project builds upon the previously designed control algorithms and experiments. It designs detailed uses cases to test RD2C-developed mitigation schemes in specific realistic scenarios, analyze and save the measurement data sets. Having a high-confidence and consistent data set is crucial for identifying relevant metrics to describe the system resilience in specific real-life-like scenarios of power grid operation.

The initial phase of the FY24 effort involved a review of current RD2C project efforts. This review and the technical discussions provided the team information need to identify that SLAC3R Cooperative Control was the next logic choice of Thrust 2-developed resilience-driven controllers for integration into the HYPERSIM IEEE 123-node Model environment. This control was understood to have been tested thoroughly in the GridLAB-D™ environment and relatively simple to implement. Additionally, this control was designed to act in a coordinated fashion, using time-scale separation, with the SLAC3R Local Control. SLAC3R Local Control had been integrated and tested during the FY23 effort. Therefore, integrating and testing the SLAC3R Cooperative Control would provide the benefit of a comparison between the system resilience improvement with both controls in operation. Last, the SLAC3R Cooperative Control features peer-to-peer communication, which would create the use case for implementing a communication system model in ns-3 so that the peer-to-peer communication links could be subjected to natural communication system disruptions as well as cyber-attacks using the ns-3 HiL capabilities.

Having selected SLAC3R Cooperative Control, this control was developed and integrated into the Opal-RT HYPERSIM model of the IEEE 123-node system to establish a platform on which scenarios and experiments could be run to gather data. The insights gained while running through many fail-fast/learn-faster cycles helped the team improve the HYPERSIM model, redefine experiment configuration and/or propose additional tests for the consequent cycles.

Similarly to the FY23 effort, the data sets acquired at the end of these simulations showed realistic performances that allowed for further investigation into how to quantitatively measure the system resilience. Performance-based metrics have continued to be researched and implemented, so that they could be applied to certain measurements and comparatively analyze the performance of the mitigation schemes towards system resilience under certain uncertainties, faults and adversary events.

This report is organized as follows. Section 2.0 details the modifications made to the HYPERSIM IEEE 123-node system model developed in FY23 as well as the justifications. Section 3.0 documents the development of a Python tool that was used to automate the running of scenario tests. Next, Section 4.0 discusses the restructuring of the SLAC3R Cooperative Control Python implementation to achieve pseudo-real-time compatibility and integration with the IEEE 123-node model. Descriptions of the scenarios used to test the SLAC3R Cooperative Control and the results including the resilience improvement assessment are provided in Section 5.0. Section 6.0 presents the challenges encountered while working with HYPERSIM to accomplish the FY24 scope. Finally, Section 7.0 summarizes the resilience assessment of the SLAC3R Cooperative Control, key takeaways from the FY24 effort, and the next steps in FY25.

2.0 IEEE 123-node HYPERSIM Model to Support Cooperative Control Testing

This section discusses the HYPERSIM model that was developed to test the performance of the SLAC3R Coordinated Control strategy. The Validation Team met with the SLAC3R Control Team to determine whether the IEEE 123-node Model that was used for testing SLAC3R Local Control could also be used to test the Coordinated Control. The SLAC3R Control Team communicated the main requirement was that the model should contain at least one microgrid capable of islanded operation and containing six GFM IBR that could be controlled using the Coordinated Control.

2.1 IEEE 123-node Model Changes to Support SLAC3R Coordinated Control Testing

The most expedient approach to obtaining a HYPERSIM model that would meet the requirement was to switch the three GFL IBRs to GFM IBRs in the IEEE 123-node Model used to test Local Control. Then the microgrid formed by connecting the three small microgrids in the IEEE 123-node Model would meet the requirement for Coordinated Control testing. The ratings of the new GFM IBRs would be the same as the old GFL IBRs, preserving the ratio of inverter-based generation to total generation in the microgrid. The new IEEE 123-node Model is illustrated in Figure 1.

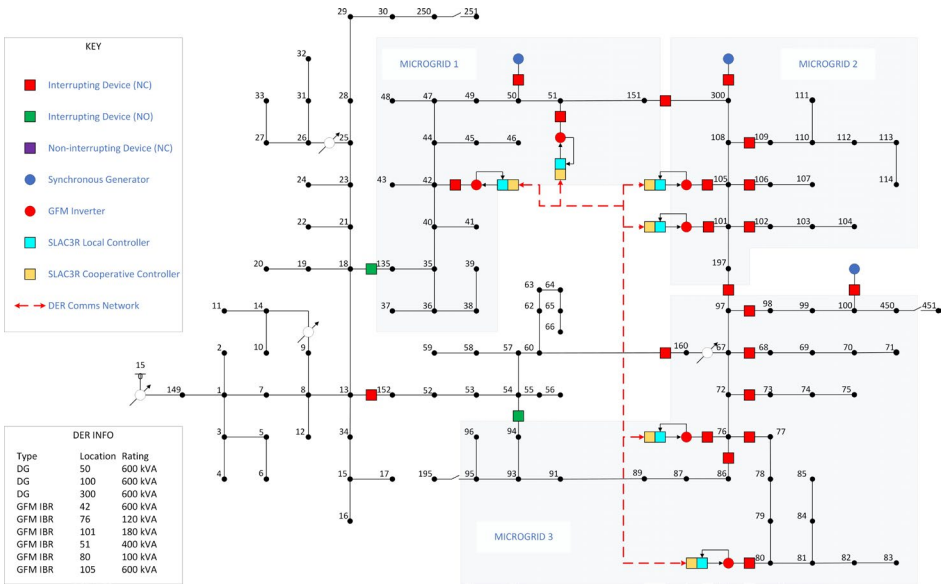


Figure 1: IEEE 123-node Model developed in HYPERSIM and used to test SLAC3R Coordinated Control.

The process for implementing the required changes was as follows.

1. Replace the GFL control with droop-based GFM control in each of the inverters located at nodes 42, 76, and 101.

2. Add the HYPERSIM model of SLAC3R Local Control to inverters located at nodes 42, 76, and 101.

3. Set all SLAC3R Local Control parameters to the values provided in Table 1.

Table 1: SLAC3R Local Control Parameters

Parameter	Value
Power-Frequency Droop, pu	0.05
a, pu	1e6
Upper Frequency Safety Threshold, Hz	60.5
Lower Frequency Safety Threshold, Hz	59.5

As with prior phases of the Validation Team's effort, a workflow was used to ensure documentation and version control of the HYPERSIM IEEE 123-node Model development (refer to Section 4.1 of the FY23 report).

2.2 GFM Inverter Passive Synchronization

A passive synchronization scheme was added to all six GFM IBR controller models to allow for more consistent, stable simulation starts. The scheme operates by measuring the grid voltage and frequency at the point-of-common-coupling (PCC) breaker. Then the grid voltage is used as the initial inverter PCC voltage reference and the initial inverter frequency is set to the sum of the grid frequency and a very small slip frequency. The scheme measures the phase angle displacement across the PCC breaker and when a sufficiently small displacement is detected, the PCC breaker is closed and the inverter voltage and frequency references are determined by droop controls. This logic is illustrated in Figure 2.

Unlike synchronous generators, governors, and excitation system models, inverter models cannot be initialized by the HYPERSIM load flow process. Thus, the IEEE 123-node Model experiences a substantial transient upon simulation start, similar to experiencing faults at each inverter model AC, three phase terminal due to the initial terminal voltages being zero. In some cases, the initial system configuration (status of loads, topological breakers, DERs, etc) can result in a starting transient that triggers sustained, unstable inverter behavior, forcing a stop to the simulation before any test can be performed. This is particularly the case for GFM IBR models because unlike GFL IBR models, the GFM control system lacks a fast, closed loop control of the inverter currents that would otherwise prevent a rapid, unrecoverable collapse of the inverter DC bus voltage. Adding the passive resynchronization scheme was critical to enabling the start of the simulation to occur with the GFM IBR PCC breakers in the open state. Then any initial network transients were allowed to dampen out prior to the IBRs performing the synchronization step and closing the PCC breaker.

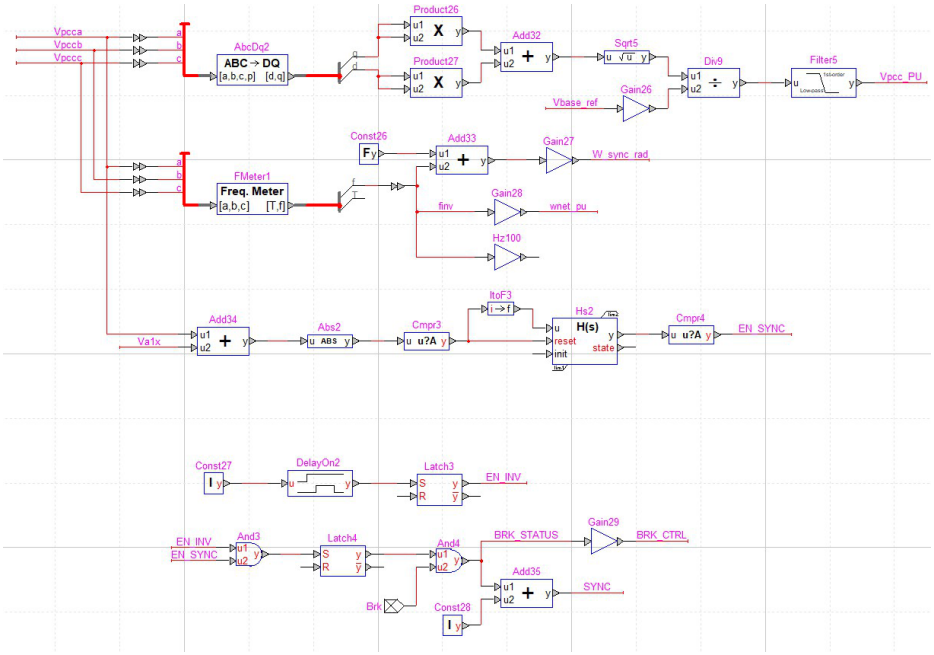


Figure 2: Inverter passive synchronization scheme.

3.0 Automation of HYPERSIM Scenario Execution

Introducing the SLAC3R Cooperative Control into the test environment represents an increase in the complexity of the inverter control systems. The inverter primary controls would operate at the lowest level of hierarchy and fastest time scale, followed by the SLAC3R Local Control and then the SLAC3R Cooperative Control. The increase in complexity is reflected in the number of different control system parameters that must be managed by the user each time a scenario is tested. This creates the need to design a scenario testing automation process that would leverage the HYPERSIM Python programming language application programming interface (API). Doing so decreases the amount of user interaction required to conduct testing of the SLAC3R Cooperative Control and therefore the chance for human error to lead to erroneous or invalid scenario test results by inadequate control of test variables.

The Python based scenario automation program has several functional requirements, which are summarized in the following list.

1. Reduce the amount of user interaction required to run a test of a given scenario.
2. Trigger the event(s) that would occur during the scenario and configure the magnitude of the event(s) (such as how much generation is lost during a generation trip event).
3. Control important model parameters to ensure certain critical parameters remain consistent between scenario tests as applicable (such as the default SLAC3R Local Control parameters listed in Table 1).
4. Configure data capture using the Datalogger feature of HYPERSIM.
5. Create a future proof framework for automating scenarios that could be maintained and expanded as additional scenarios with different events are conceived and pursued.

There are two main approaches to achieving the second functional requirement above (the remaining requirements can be achieved using either approach).

1. The Python script is designed to trigger events by making API calls to change component parameter values *while the simulation is actively running.*
2. The Python script is designed to trigger events by making API calls to change component parameter values *before the simulation is started.*

The serious issues with the HYPERSIM Python API, uncovered during the FY24 effort and documented in Section 6.3, precludes using the first approach. In contrast, the second approach guarantees the integrity of the numerical integration method used to perform the real-time simulation. However, the second approach also requires additional modeling effort to instantiate the mechanisms to control events using built-in HYPERSIM components, which is described in the next section.

3.1 HYPERSIM Model Changes to Support Non-API Triggered Events

Instead of making a HYPERSIM Python API call at a predetermined time to change component parameters, built-in HYPERSIM components are used to create logic circuits that control the

Figure 3: Logic circuits to cause a signal transition at a specific simulation time from (a) zero to non-zero and (b) one non-zero value to another non-zero value.

timing and value of component inputs. Two examples of logic circuits are shown in Figure 3.

Figure 3a shows a circuit that controls whether Inverter 42 trips offline during the simulation. If the Integer Constant block value is one, then a delayed pulse will occur at the output of the Pulse Delay block. The pulse width may be set to a large number to create a permanent change in the Pulse Delay block output. The HYPERSIM Python API can be used to set the value of the Integer Constant and the delay value of the Pulse Delay block, thus causing a transition from zero to one to occur after a time lapse, equal to the pulse width, from the start of the simulation.

Figure 3b shows a similar circuit, except that the Pulse Delay block will control whether input u1 or u2 will be passed through to the output y of the Selector block. Thus, the API may be used to control a transition between two non-zero values that occurs after a time lapse, equal to the pulse width, from the start of the simulation. The logic circuit components should have recognizable names relating to the event being configured and the function of the component. For example, the component name *T_INV_TR42* is meant to convey that the component sets the trip time of Inverter 42.

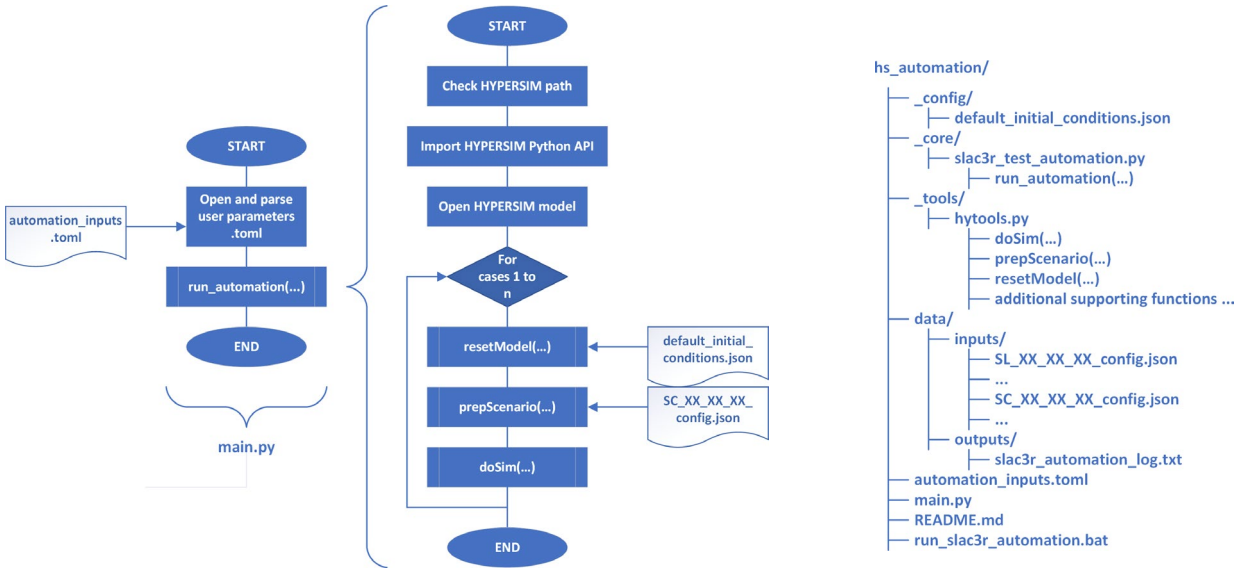
The additions of the logic circuits clearly provide mechanisms to control the timing of events, such as changes to inverter PCC circuit breaker states or input setpoints of control systems, while only making API calls prior to the simulation start. Thus, all events occurring in any given scenario, per the descriptions, are translated to separate logic circuits that are added to the HYPERSIM IEEE 123-node Model. Then, depending on the scenario being tested, the scenario automation Python script will configure the relevant logic circuit before simulation start.

3.2 HYPERSIM Scenario Automation Python Script

According to the functional requirements outlined in Section 3.0 and the event triggering approach presented in Section 3.1, a Python script was developed as part of the SLAC3R Cooperative Control testing effort. The program execution and structure are described in the pseudo code flowchart shown in Figure 4.

The scenario automation Python script accessed three types of input files.

1. *automation_inputs.toml*
2. *default_initial_conditions.json*



(a) high-level, pseudo-code flowchart of the program execution (b) script file structure

Figure 4: The HYPERSIM scenario automation Python script.

3. *SC_XX_XX_XX_config.json*

The purpose of the *automation_inputs.toml* file is to provide the user the ability to set local system dependent file paths as well as to point to the scenario configuration files that would be accessed to run one or more specific scenarios.

The *default_initial_conditions.json* contains the names of model blocks and their internal parameters and is accessed by the *resetModel* function. This allows the program to reset all relevant parameters such that a consistent initial condition is achieved for each scenario test including disabling all event triggers.

The *SC_XX_XX_XX_config.json's* files contain the block names and parameter names to modify the model parameters to perform simulations of specific scenario tests. The "SC" prefix indicates a SLAC3R Cooperative Control test scenario and "XX" is a placeholder for the specific scenario, experiment, and trial codes per the file naming convention. These files are accessed by the *prepScenario* function. Executing the *prepScenario* function includes overwriting some model parameters settings in the *default_initial_conditions.json* in order to schedule the specific events needed for a given scenario. Then the simulation is started, data is collected, and the simulation is stopped during execution of the *doSim* function without requiring a single function call by the HYPERSIM API while the simulation is running.

To further reduce user interaction, the *run_slac3r_automation.bat* file allows the user to run a single shell script that precludes the need for any particular IDE software to run the *main.py* file. The *run_slac3r_automation.bat* need only be modified once to point to the location of the correct Python interpreter included with the HYPERSIM installation.

3.3 HYPERSIM Scenario Automation Results

In summary, the HYPERSIM model development work described in Section 3.1 and Python programming work described in Section 3.2 greatly reduced the effort needed to run multiple trials of multiple scenarios testing the SLAC3R Cooperative Control. A framework that is agnostic to both scenario and control-under-study was achieved. At the time of this report, the HYPERSIM scenario automation Python script supports several types of events as listed here.

1. Inverter trip offline,
2. Synchronous generator trip offline,
3. Unplanned islanding,
4. Distribution system network fault and fault clearing at Node 93,
5. Unplanned load shedding.

Additionally, the program supports control of the following groups of model and simulation parameters.

1. Inverter active power dispatch,
2. Synchronous generator governor speed bias (sets active power output),
3. SLAC3R Local Control enables,
4. SLAC3R Local Control parameters,
5. SLAC3R Cooperative Control enables,
6. Datalogger signal group recording start times.

The framework development readily supports control of additional events and model parameters as they may become needed for investigating new scenarios and new mitigation strategies in the future.

4.0 Integration of SLAC3R Cooperative Control

The Validation Team coordinated with the SLAC3R Team to get access to the Python implementation of the SLAC3R Cooperative Control. In its original form, the Python implementation of the control was structured to be a Python federate of a HELICS co-simulation, which included a GridLAB-D model of the IEEE 123-node Distribution System to simulate the physical network dynamics. The OPAL-RT simulators are designed to be real-time digital simulation platforms, which is fundamentally different than the non-real-time, co-simulation platform that HELICS was designed to provide. Therefore, some additional effort would be required to integrate the control. Multiple options for how to integrate the SLAC3R Cooperative Control with the HYPERSIM IEEE 123-node Model were considered as presented in the following list.

1. Translate the Python code implementing the control to built-in HYPERSIM control system blocks (similar to how SLAC3R Local Control was integrated).
2. Restructure the Python code to utilize the HYPERSIM Python API to implement the communication between the OPAL-RT simulator and the Python program running the control.
3. Restructure the Python code to utilize Ethernet/IP based communication to implement the communication.

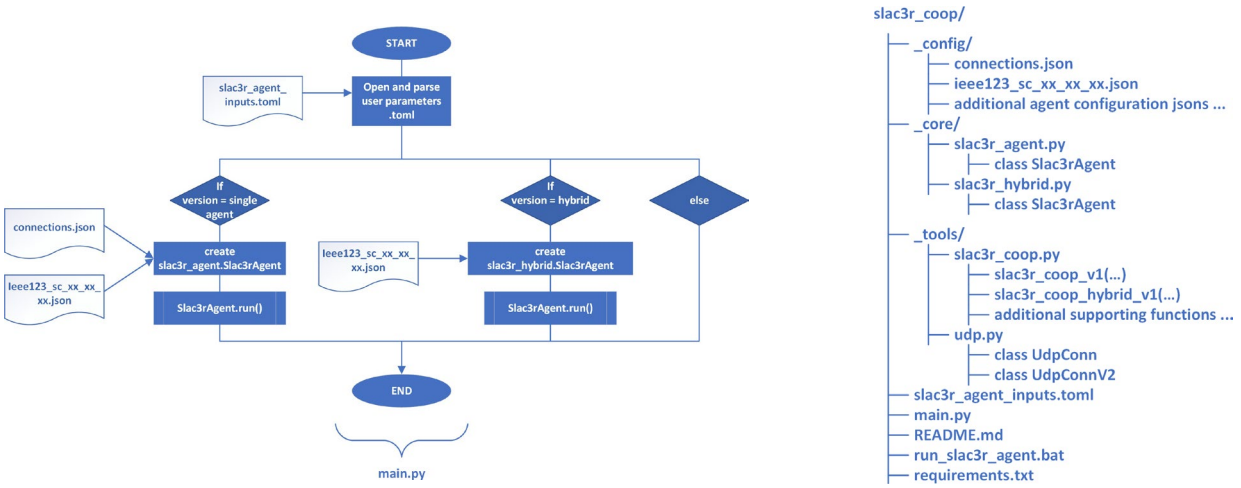
The first option was not ideal because it would require extensive modeling work using the HYPERSIM UI, which is a time intensive process. Furthermore, the issue of how to instantiate cyber-attacks when the communication links are effectively emulated with the HYPERSIM model poses a problem. Last, this approach does not provide the opportunity to develop a framework to integrate Python-based controls running on Cybernet VMs with the HYPERSIM model.

The second option was deemed infeasible due to the documented issue of HYPERSIM Python API calls causing stretched time steps as discussed in Section 6.0. Furthermore, the latency with which the API calls are processed would prevent implementing the control with the minimum desired update rate of 0.5 s.

This left option three, which was considered the best option for the reason that the issues highlighted above could be avoided. Initially, it was decided to implement the communication links using the Python socket library. The library provides the classes and methods needed to create UDP streams that would connect the Cybernet VM running the SLAC3R Cooperative Control to the OPAL-RT simulator. The UDP streams transmit data in the payloads of IP packets across the Cybernet Under Network (a virtual partition of the LAN used to connect all of the Cybernet OT and IT devices). Programming UDP streams is straightforward and the Python library is well documented. At a later date, the test environment can be improved by using industry standard protocols such as Modbus-TCP and DNP3-TCP.

A two stage process was adopted for integrating the controls to achieve an incremental approach. The first stage is documented in Section 4.1 followed documentation of the second stage in Section 4.2. A flowchart of the *main.py* script, which implements either version depending on the user inputs is illustrated in Figure 5.

The Python code delivered by the SLAC3R Team was parsed to extract the SLAC3R Cooperative Control implementation. That code was refactored to provide the calculated IBR active power setpoints as a callable function *slac3r_coop_v1*, which is defined in the



(a) high-level, pseudo-code flowchart of the program execution (b) program file structure

Figure 5: The SLAC3R Cooperative Control Python program.

slac3r_coop.py file. In the second stage, when the Python implementation of the cooperative control math needed to be made flexible to handle an arbitrary subset of the GFM IBRs, a second callable function *slac3r_coop_hybrid_v1* was created and added to *slac3r_coop.py*.

4.1 SLAC3R Cooperative Control Integration Stage One - Single Agent

In the initial stage, the cooperative control code was restructured to achieve a single instance of the program, referred to as a SLAC3R agent, running on a Cybernet VM. The single agent controlled all GFM IBRs in the simulation through the closed-loop, UDP communication described in the prior section. This avoided having to modify the Python implementation of the math underlying the control and allowed the Validation Team to focus on coding the UDP interfaces. In this version, the peer-to-peer communications are effectively baked into the program that implements a single SLAC3R agent. Figure 6 shows the physical and logical connections used to integrate the single agent version of SLAC3R Cooperative Control with the IEEE 123-node Model.

A single agent is instantiated by configuring the UDP connection with the OPAL-RT simulator using the *connections.json* file. Then the SLAC3R Cooperative Control parameters for that agent are specified in a scenario specific configuration file, which has a file name pattern of *ieee123_sc_xx_xx_xx.json*. The “sc” suffix indicates a SLAC3R Cooperative Control agent configuration and “xx” is a placeholder for the specific scenario, experiment, and trial codes per the file naming convention (see Section 4.1 of the FY23 report). The actual operation of the agent is defined by the *run* method of the *slac3r_agent.Slac3rAgent* class, which is described using a flowchart of pseudo-code in Figure 7.

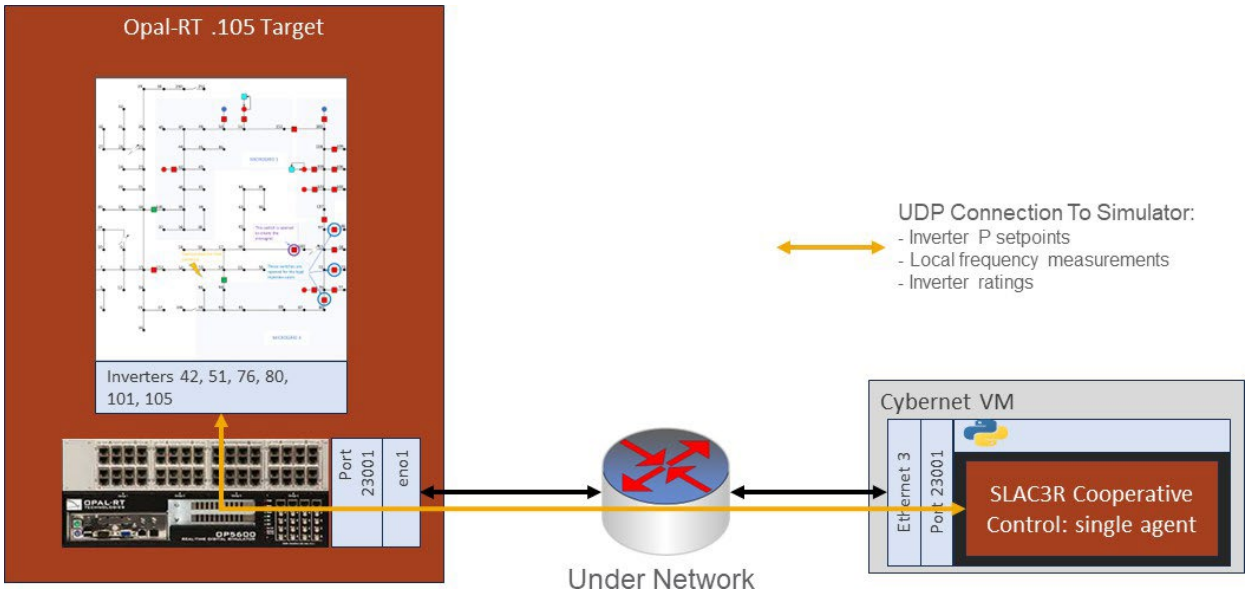


Figure 6: Integration of single agent version of SLAC3R Cooperative Control with the IEEE 123-node Model running on an OPAL-RT simulator.

4.2 SLAC3R Cooperative Control Integration Stage Two - Hybrid Agents

The hybrid version of SLAC3R Cooperative Control was developed to allow testing of the control under two conditions.

1. The peer-to-peer communication links would be exposed to either real or emulated IT network latencies.

2. The control would be implemented on a fewer number of VMs than GFM IBRs modeled in HYPERSIM.

This required restructuring the single agent Python code to allow a flexible number of UDP connections with peer agents running on separate VMs. Additionally, the Python implementation of the SLAC3R Cooperative Control math needed to be reworked to permit calculating the power setpoints for an arbitrary subset of GFM IBRs using a combination of prior power setpoints from local variables and prior power setpoints from the peer agents. Figure 8 shows the physical and logical connections used to integrate two hybrid agents with the IEEE 123-node Model.

Each hybrid agent is instantiated on a separate VM using a single JSON file with name pattern of *ieee123_sc_xx_xx_xx.json*. In this case, the JSON file defines both the SLAC3R Cooperative Control parameters and the UDP connection information for both connections to the OPAL-RT simulator and connections to peer SLAC3R agents. To run a test of the control, the user would ensure the Python program is installed on a minimum of two Cybernet VMs. Then the user would create separate and unique JSON agent configuration files for each instance on the separate VMs. The operation of a hybrid agent is defined by the *run* method of the *slac3r_hybrid.Slac3rAgent* class, which is described using a flowchart of pseudo-code in

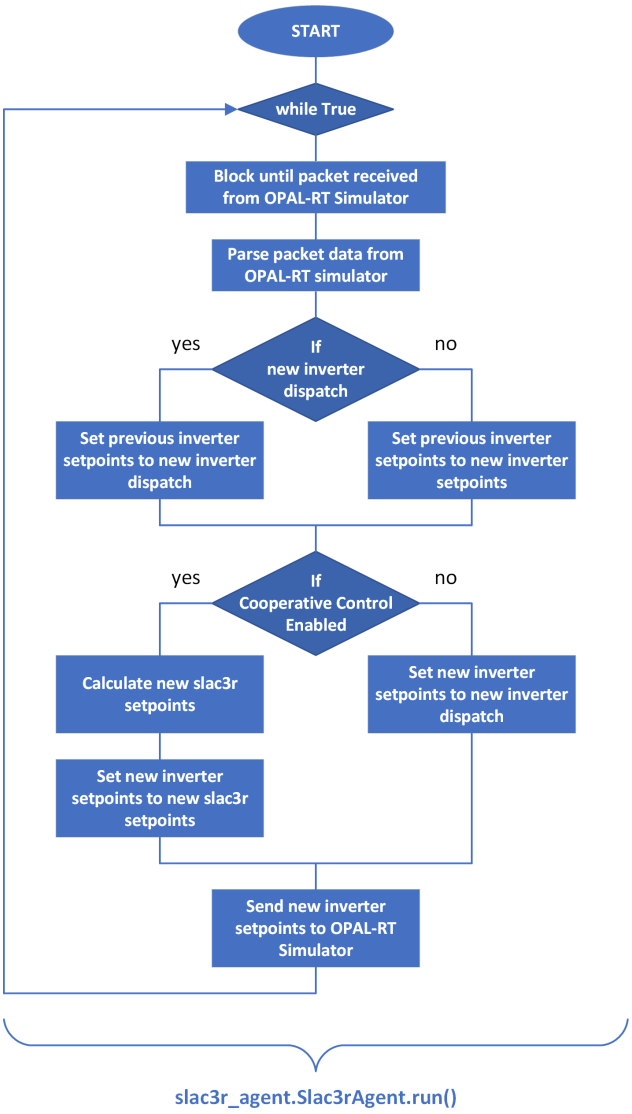


Figure 7: Single agent version of SLAC3R Cooperative Control code execution.

Figure 9.

4.3 Achieving Pseudo-Real-Time Implementation of SLAC3R Cooperative Control

The Windows operating systems (OS) that are virtualized on the Cybernet VMs are not real-time operating systems. The result is that there is no mechanism to guarantee that each iteration of the infinite while loops described in the SLAC3R Cooperative Control Python programs (see Figure 7 and Figure 9) complete within a specified time frame (as would be the case with a real-time OS implemented on a PLC or DSP chipset). To overcome this limitation, the Validation Team leveraged the fact that the OPAL-RT simulator can be configured to publish UDP packets with a predetermined publication rate.

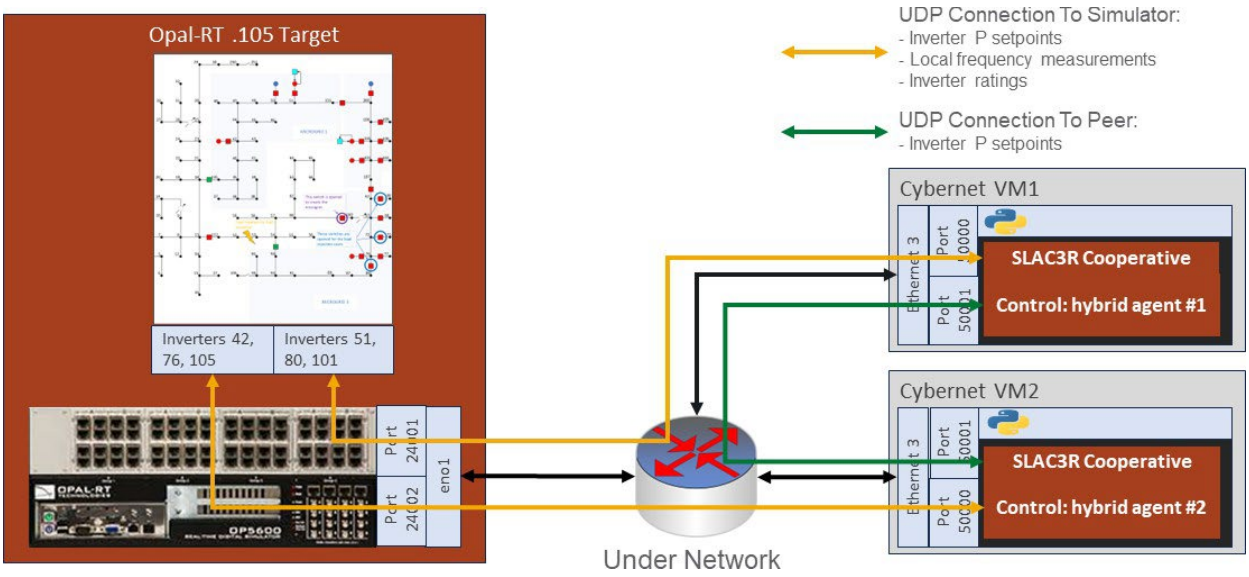


Figure 8: Integration of the hybrid agent version of SLAC3R Cooperative Control with the IEEE 123-node Model running on an OPAL-RT simulator.

The UDP sockets implemented in the control code for receiving data from the OPAL-RT simulator were configured for blocking operation with a long timeout period. This causes control code execution to pause each cycle until a packet from the OPAL-RT simulator is read from the buffer. If the timeout duration is exceeded then it is likely that the simulation has stopped running on the simulator. So long as the control code single cycle execution time is significantly less than the simulator publication rate, the control code execution can be synchronized with the OPAL-RT simulator. Under this paradigm, the control system effectively operates in pseudo-real-time fashion.

This methodology can be extended to hierarchical control systems by configuring a separate UDP stream for each control system implemented on a Cybernet VM. Then the publication rate for each stream can be configured to match the intended cycle time for the corresponding control. If no data exchange is required between control system and the OPAL-RT simulator, then the UDP stream simply acts as a heartbeat, which also functions to let the control system know that the simulation is in progress.

4.4 SLAC3R Cooperative Control Integration Results

In summary, the Validation Team produced a workable, pseudo-real-time implementation of the SLAC3R Cooperative Control that can be integrated with a HYPERSIM power system simulation. The implementation is agnostic to the specific model topology so long as the UDP configuration in the OPAL-RT simulator (IP settings and packet payloads) is coordinated with the SLAC3R agent configuration JSON files. Furthermore, the control code was implemented using a modular code structure that can be adapted for other control systems. Due to time and budget constraints on the Validation Team's efforts in FY24, only two scenarios were tested in

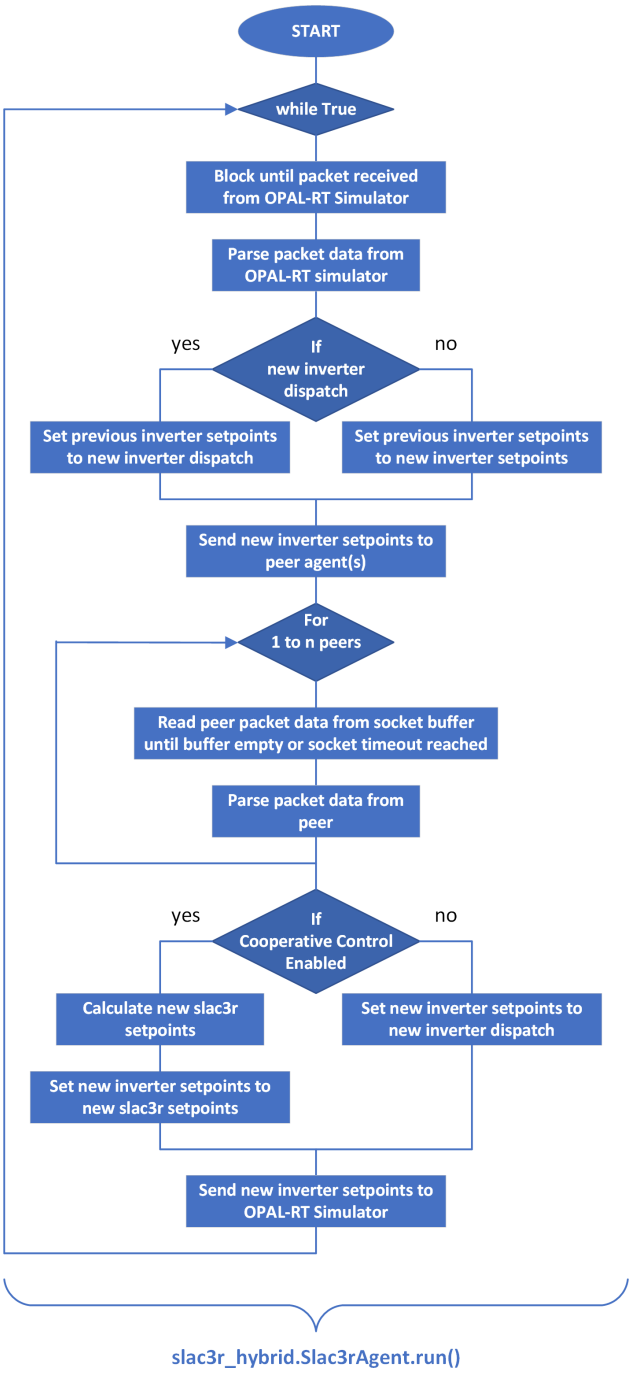


Figure 9: Hybrid agent version of SLAC3R Cooperative Control code execution.

the IEEE 123-node Model using the two hybrid agent implementation shown in Figure 8. Descriptions of the scenarios and the results are presented in Section 5.0.

5.0 SLAC3R Cooperative Control Testing and Resilience Assessment

5.1 SLAC3R Cooperative Control Configuration

The SLAC3R Cooperative Control features both control system parameters and a connection matrix that defines how the control cooperation will be performed. Both aspects of the control system configuration should be recorded and associated with the test data that was generated by simulating the control system. Due to time constraints and unexpected challenges in the model development phase of the FY24 effort, only one configuration of the SLAC3R Cooperative Control was formally tested using the scenario and experimentation procedures developed in FY23.

The tests of the SLAC3R Cooperative Control presented in Section 5.3.1 and Section 5.3.2 were performed using the hybrid agent implementation of the control system. The hybrid implementation is described in Section 4.2 of this report. The control configuration is captured in the JSON configuration files used to configure the two hybrid agents that were instantiated on two separate Cybernet VMs. Per the agent configuration files, Table 2 provides the control system parameters used by both agents.

Table 2: SLAC3R Cooperative Control Parameters

Parameter	Value	Parameter	Value
Nominal Frequency, Hz	60	Default Power Reference (a), pu	0.0
η, pu	0.5	Frequency Bias	0.5
Power Bias	0.5	e	1.0
Droop Slope, pu	0.05		

(a) The SLAC3R Cooperative Control uses this value if no inverter active power references are received from the centralized DER dispatch control system.

The connection matrix defines which inverter controllers would communicate to facilitate the real-world implementation of this distributed control system. However, due to time constraints the hybrid agent version of SLAC3R Cooperative Control was developed to facilitate a having a fewer number of hybrid agent instances emulate the peer-to-peer communication links indicated by the connection matrix. Thus, each hybrid agent is configured with knowledge of the connection matrix as well as the row and column indices corresponding to the specific inverters that the agent controls. Thus, the agent uses the connection matrix and the row and column indices to know which power references to expect from the peer hybrid agent (communicated over a UDP connection) and which power references to expect from local variables. The logical connections being emulated by the hybrid agents for the IEEE 123-node Model discussed in Section 2.0 is shown in Figure 10.

The connection matrix defines the Peer-to-Peer connections. The details of how to parameterize the matrix are left to the documentation of the SLAC3R controls. Instead, the connection matrix

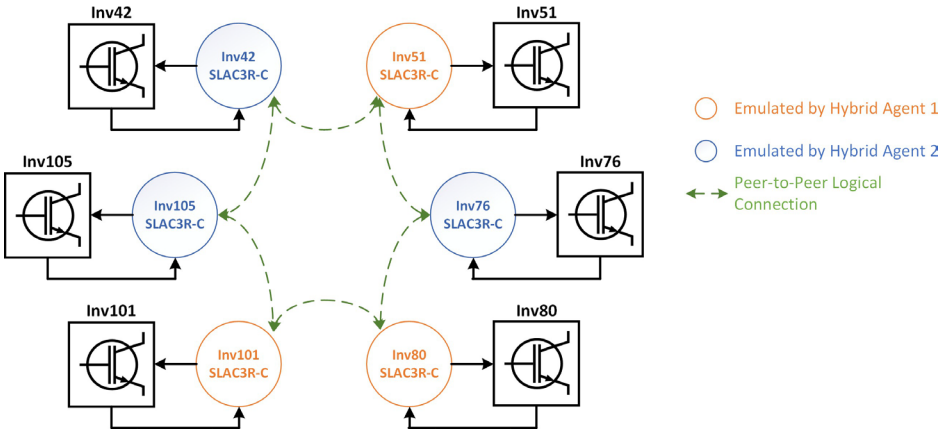


Figure 10: Logical connections between the SLAC3R Cooperative Control agents being emulated by the hybrid agents instantiated on the Cybernet VMs (see Figure 8).

used for the testing of the control presented in the following sections is given below.

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \quad (1)$$

In the connection matrix, the row and column indices correlate to the node numbers of the inverters in order of ascending node number. Thus, Inverter 42 corresponds to row one and column one. The correlation ends with Inverter 105, which corresponds to row six and column six.

5.2 Resilience Assessment

To assess the overall power system resilience to unplanned adversarial actions when engaging resilience-driven mitigation strategies, performance-based metrics have been identified and applied to quantitatively measure the controller performance. Given the currently tested mitigation strategies, the power grid assets they act upon (the IBRs), and the main scope of their controlling actions, there are two indexes according to which resilience is going to be analyzed:

- Ability of the IBRs to provide frequency support to the microgrids during the disturbances,
- Ability of the IBRs to alleviate synchronous legacy generators effort to accommodate for sudden demand variation due to unforeseen events.

By their definition, these indexes are related to certain system measurements, also named, as detailed in the FY 2023 report, figures of merit or measures of performance. Specifically, the frequency of the synchronous generators is an indicator of how balanced the system is when it comes to serving the demand and avoiding load shedding. At the same time, the power generated by the distributed energy resources, that is the IBRs in this case, gives information about how capable the demand can be sustain by local DERs, rather than spinning new synchronous generator reserves.

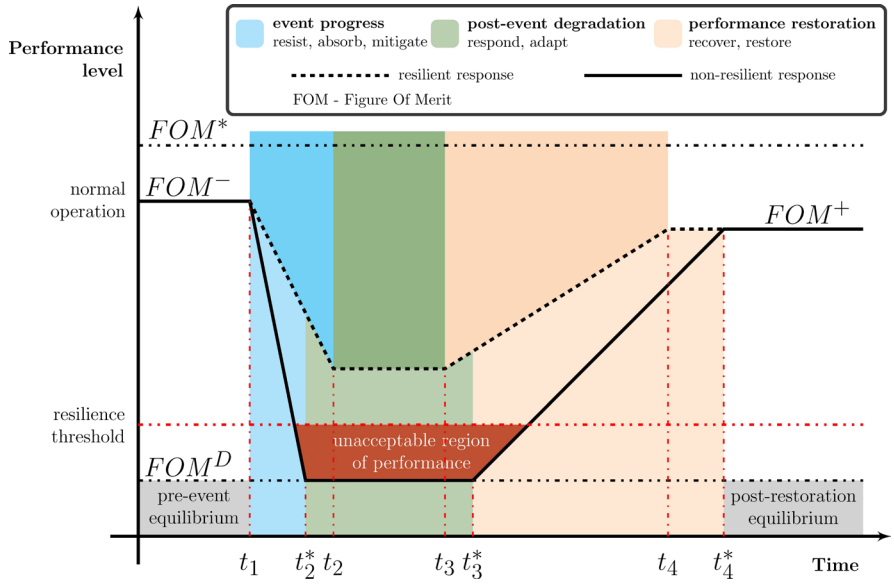


Figure 11: Evolution of performance indicator (FOM) of a system in the presence of event (*resilience curve*).

Moreover, in an attempt to quantitatively measure the system resilience, a set of metrics have been applied to the synchronous generators frequency measurements. Currently, the computed metrics, mainly defined on frequency as FOM according to Figure 11, though applicable to any signal, are

- Rate of Change of Frequency (RoCoF), that is the slope of the line tangent to the frequency curve right after the event

$$|\dot{\omega}| = \frac{|\omega(t_0 + T) - \omega(t_0)|}{T},$$

  where $t_0 > 0$ is the time when the disturbing event happens, and $T > 0$ is the RoCoF calculation window

- Maximum frequency deviation $\Delta\omega_{max}$ from the initial stable/nominal value, also known as frequency nadir/zenith

$$\Delta\omega_{max} = \max_{t \geq t_0} |\omega_0 - \omega(t)|$$

- Degradation time, that is the amount of time it took for the system to degrade, absorbing the disturbance

- Recovery time, that is how long it took for the system to recover to a stable condition
- Vulnerability index (VI) [1], which evaluates the level of degradation

$$VI = \frac{|\omega_0 - \omega_{\min / \max}|}{\omega_0}$$

A VI value of 0 puts the system in a perfect condition, while a value of 1 means the system has completely degraded.

- Normalized degradation index (DI) [1], which gives information about the temporal behavior of the grid during the degradation stage

$$DI = \frac{\int_{t}^{t_2} (\omega_0 - \omega(t))\, dt}{\omega_0 (t_2 - t_1)}$$

A zero DI implies a fully resistant system that suffers no degradation during the event, while a value of 1 would render the system completely nonfunctional.

- Restoration efficiency index (REI) [1], as the measurement of the efficiency in the restoration process

$$REI = \frac{\int_{t}^{t_4} (\omega - \omega_{\min / \max})}{|\omega_0 - \omega_{\min / \max}| (t_4 - t_3)}$$

A system with a full capacity of restoration should have an $REI = 1$, while one which would fail to recover will have $REI = 0$.

- Recovery index (RI) (resilience index in [1]), which informs about resilience by looking at the entire dynamics from the beginning of FOM degradation to the end of the restoration period.

$$RI = \frac{\int_{t_1}^{t_4} \omega(t)\, dt}{\omega_0 (t_4 - t_1)}$$

The closer to one the RI is, the more resilient the system.

5.3 Exemplification Scenario Descriptions

Two scenarios are considered in this section to analyze how the SLAC3R strategy mitigates the system resilience against possible natural or cyber threats. These scenarios are presented in the following sections. For each scenario, three use cases are considered:

- base case - when the system dynamics are observed before and after the unforeseen event without engaging any resilience-driven controller,
- local SLAC3R - when the local SLAC3R controller is engaged to ensure the system frequency is constraint within a safety region defined by lower and upper limits,
- local and cooperative SLAC3R - when the local safety SLAC3R controller are set to communicate with each other to reach a consensus about how much power to generate in order to restore the system frequency to nominal and serve the demand.

The self-aware local autonomous part of the SLAC3R controller has the same parameters for both exemplified scenarios, as presented in Table 1.

5.3.1 Scenario 1 - Unforeseen substation islanding

In this scenario it is assumed that while in grid connected mode, a hypothetical cyber attack causes the substation/microgrid POI breaker (between nodes 60 and 160 in Figure 1) to open without warning. That event leaves the distribution system relying solely on its synchronous generators and GFM IBRs to sustain the power demand. To increase the severity of this scenario, the pre-event dispatch of all GFM IBRs was set to zero. This resulted in significant energy import from the grid, which was subsequently lost when the cyber attack occurred. This could reflect a scenario where the adversary had gained significant visibility into the power system operation allowing them to coordinate the attack with a period of heightened vulnerability to unplanned islanding.

The goal of the experiment is to determine the system response to unintentional islanding that is caused by malicious activity (i.e. not protective action). Based on the frequency and power measurements under the three use cases, the ability of the SLAC3R controller to maintain and improve system resilience is going to be quantified through metrics in Section 5.2.

A sudden disconnection of the distribution system from the main grid would result in a very high drop in frequency, as seen in the base case in Figure 12, ending with significant loading of local DER. Hence the system frequency stabilizes on a non-nominal value due to the linear relationship between active power output and system frequency in the droop controls implemented in the DER control systems. Engaging the SLAC3R local control in the 6 DERs represented by GFMs manages to rapidly redispatch the GFM IBRs to supply more active power, such that the frequency is brought inside the safety region defined by the upper and lower limits in Table 1, specifically very close to the lower threshold. However, the nominal frequency is still not achieved. This last challenge is taken over by enhancing SLAC3R with the coordinated/cooperative part of the controller, as described in Section 5.1. This way a full recovery of the system is achieved with frequency being brought back to nominal.

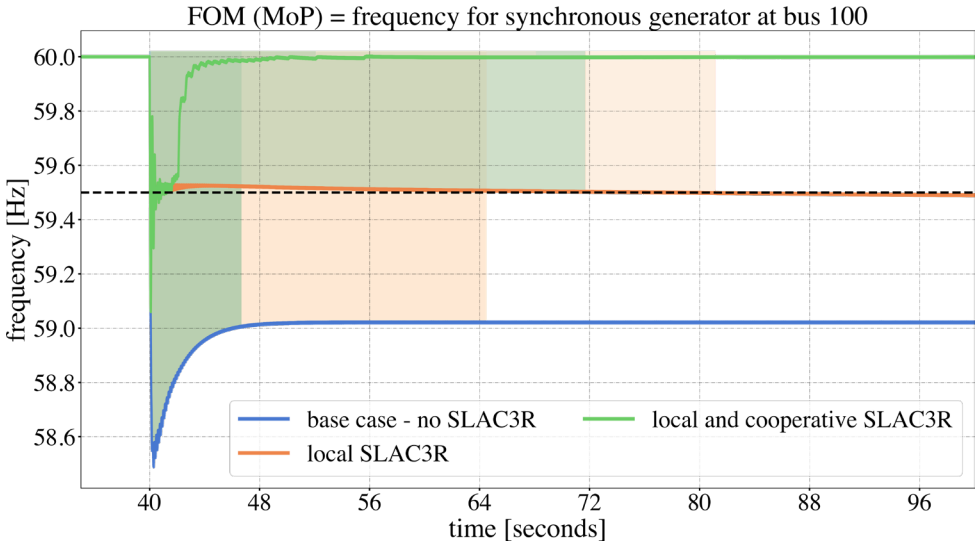


Figure 12: Scenario 1 - synchronous generator frequency

As mentioned above, the system frequency recovery is a consequence of controlling the DERs to adequately redistribute the energy generation to sustain the full demand. Figure 13 shows

the total generation the IBRs are capable of in the 3 use cases. It is clear to see that when the IBRs find a way to cooperate by letting neighbors know about their actions, they can achieve a consensus on how much power to produce extra in order to meet the full power demand.

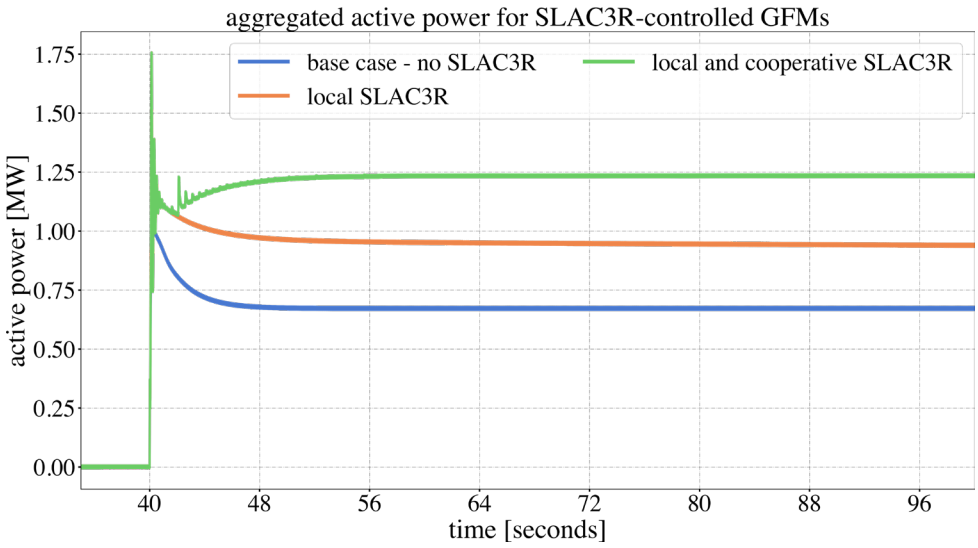


Figure 13: Scenario 1 - aggregated GFMs/IBRs power output

With the frequency measurements in Figure 12 as FOMs, the resilience metrics previously presented are normalized and plotted as a spider/radar graph in Figure 14. It can be concluded clearly from the radar graphs numbers that the third use case when both the local and cooperative SLAC3R mitigation strategies are engaged is the most resilient case when the system gets islanded. The normalized REIs and RIs are 1, while the maximum deviation from nominal, and degradation and recovery times are small.

5.3.2 Scenario 2 - DER loss while islanded

In this scenario it is assumed that the distribution network comprised of the three microgrids in the IEEE 123-node system has been intentionally disconnected from the main grid by opening the breaker between nodes 60 and 160 (see Figure 1). This causes the system to rely on the local synchronous generators and GFM IBRs to sustain its own power demand. The use case for an intentional islanding operation could be to insulate the local loads from active or anticipated disturbances in the larger distribution system in which the microgrids are situated. Then, while in islanded mode, the system experiences the loss of the IBR at node 42 in Figure 1, leading to a significant loss in generation, that is approximately 600 KW. The hypothetical real-world situation this scenario is to emulate could be either equipment malfunction or cyber attack on a DER's PCC circuit breaker control system. Once again, in the case of a cyber attack, this scenario illustrates how an adversary could use visibility of the microgrid operations, namely the status of the POI circuit breaker, to coordinate an attack when the system would be most vulnerable to loss of generation or energy storage resources.

The goal of this experiment is to determine the system response to the 600 KW contingency from the IBR at node 42 while in islanded mode and observe if integrating the SLAC3R

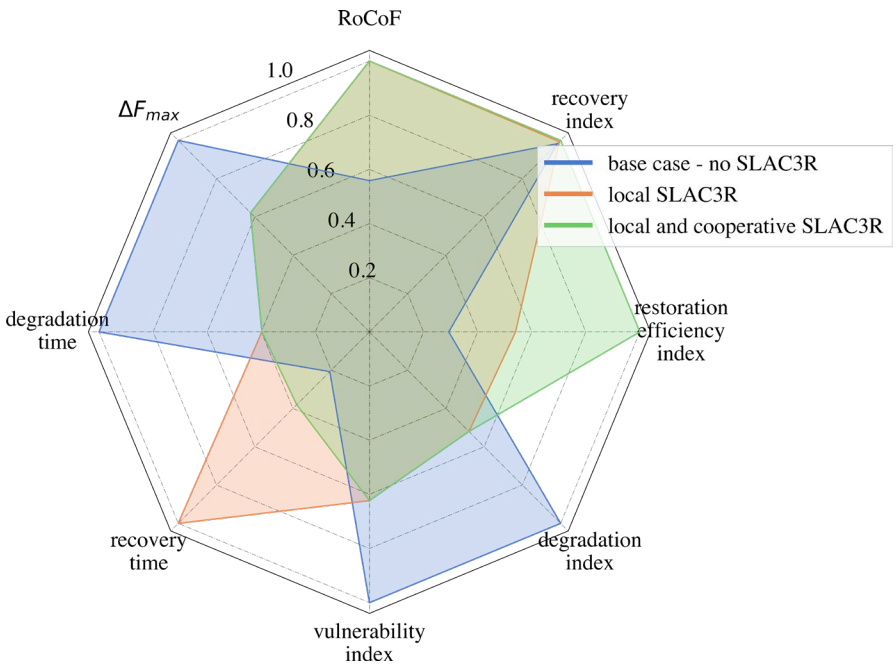


Figure 14: Scenario 1 - synchronous generator frequency metrics for resilience

cooperative control improves the resilience metrics.

Similar conclusions to those in the case of Scenario 1 in Section 5.3.1 can be drawn for the use cases of this scenario as well, with the visuals in Figures 15, 16, and 17.

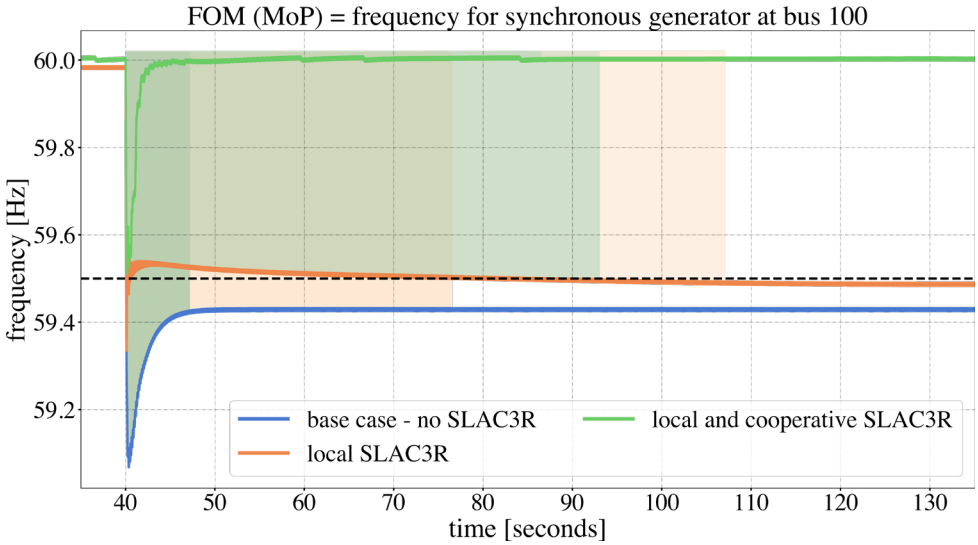


Figure 15: Scenario 2 - synchronous generator frequency

These two scenarios exemplify how cooperation between the IBRs controlled by the SLAC3R controller improve the system resilience, by not only allowing the system to register the event

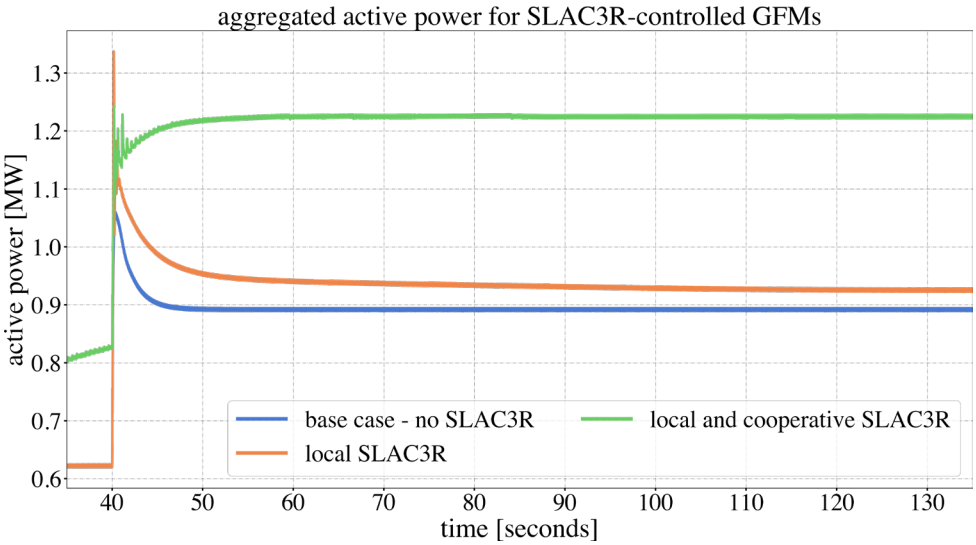


Figure 16: Scenario 2 - aggregated GFMs/IBRs power output

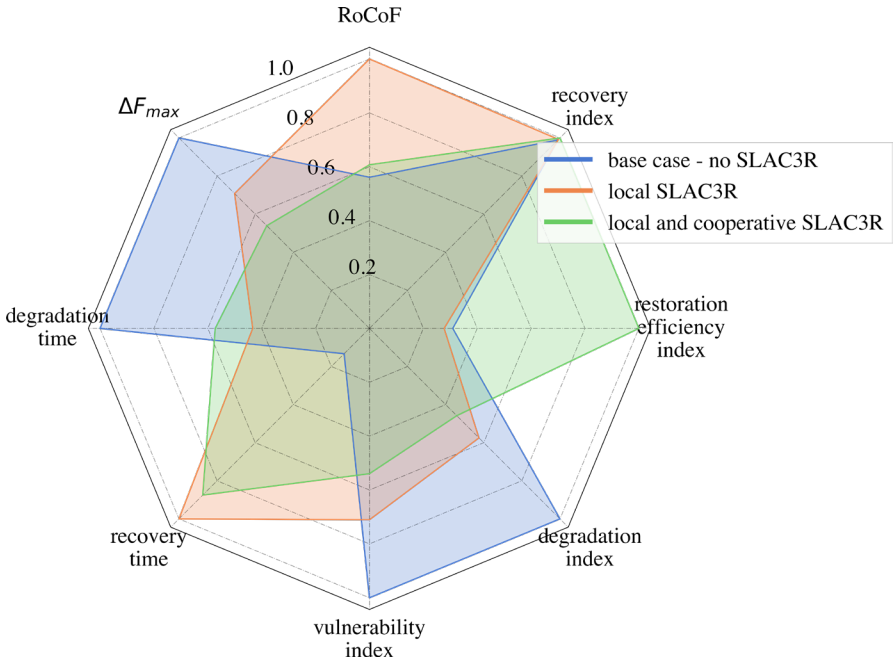


Figure 17: Scenario 2 - synchronous generator frequency metrics for resilience

faster and respond to it, but also providing enough distribution of power to assist in recovering the system to its state before the event. While the SLAC3R local safety control drives the IBRs such that they supply power to bring the system frequency within a safe region of operation, the cooperative part will push further such that the frequency is returned back to the nominal 60 Hz.

CAVEAT: Though the goal and expectation are that the restoration and recovery times are the shortest in the cases when the SLAC3R controller is engaged, due to the output signal wave form, the ripples sustained in the steady-state and the algorithms that filter the ripples to compute the mentioned times, they turn out to be larger than in the base case. This aspect remains to be investigated further to improve the resilience analysis.

6.0 HYPERSIM Challenges Encountered

This section of the report presents in detail several HYPERSIM related issues that required significant effort by the Validation Team to address in the course of further developing the HYPERSIM IEEE 123-node Model and testing the SLAC3R Cooperative Control. The issues span bugs with the HYPERSIM user interface (UI) software, unsatisfactory behavior of built-in HYPERSIM models, and unsatisfactory performance of the platform as related to fundamentals of real-time digital simulation. The following subsections discuss the issues encountered and the eventual resolutions.

6.1 HYPERSIM Bugs Encountered During Version Migration

During the initial phase of the SLAC3R Coordinated Control testing effort, the Validation Team coordinated with the Scalable Hardware in the Loop Federated Testbeds (SHiFT) Team to increment the OPAL-RT simulator HYPERSIM version. The benefits of moving to a newer HYPERSIM version were anticipated to be:

1. Access to new built-in models including PNNL developed single-phase dynamic load models,
2. HYPERSIM bug fixes,
3. Better model support from OPAL-RT by using current HYPERSIM versions,
4. Access to new HYEPRSIM user interface features.

Initially, the decision was made to migrate to a 2023.3 version of HYPERSIM. This is the latest minor revision of the second-to-most recent major revision (2024 being the latest major revision). This decision was based on the assumption that the 2023.3 would be more mature than 2024.1 and thus would feature fewer bugs. This required installing the new HYPERSIM version on both the Cybernet VM's used to run HYPERSIM as well as the OPAL-RT OP5700 simulator that would be used to run real-time simulations of the IEEE 123-node Model for testing SLAC3R Coordinated Control. Two sub-versions of HYPERSIM were tested: 2023.3.1.o35 and 2023.3.0.o26.

Both of the newer versions of HYPERSIM featured a significant bug that delayed the IEEE 123-node Model development process with the first recorded instance occurring on February 19. The symptoms of the bug were that upon attempting to start a simulation of a HYPERSIM model (whether in offline mode using the localhost or real-time mode using the OPAL-RT simulator) an error message would appear on screen, which is shown in Figure 18.

After a discussion with OPAL-RT support on March 11, it was revealed that the HYPERSIM UI works by starting multiple parallel services, which communicate by opening TCP ports. OPAL-RT support suggested that the process of selecting ports and managing TCP connections between the services is not very robust. In the case of the HYPERSIM 2023.3 versions installed on the HYPERSIM access VMs, this problem was persistent and presented a major blocker to further model development. Around the same time, another issue was observed where the HYPERSIM model analysis and model save processes would hang for long periods of time (in excess of 1 hour) and eventually fail with the message displayed in Figure 19.

Figure 18: Screenshot of HYPERSIM error message indicating a failed simulation start due to bugs in the management of HYPERSIM TCP connections.

Figure 19: Screenshot of HYPERSIM error message indicating a failure in the HYPERSIM Unicon service.

The recommendation from OPAL-RT support was to further upgrade the HYPERSIM version to 2024.1. The rational being that the TCP connection management was more robust and an error logging feature was added in 2024.1 that would allow users to easily collect all relevant HYPERSIM UI process logs such that they could be sent to OPAL-RT support.

The Validation Team elected to increase the HYPERSIM version to 2024.1.0.o39 on all of the HYPERSIM access VMs starting on March 12. This proved to be the only reliable fix to the bugs referenced in Figure 18 and Figure 19. Evidence for the issue being resolved in the newer version was found in new warning messages displayed in the HYEPRSIM console during the model analyze process as shown in Figure 20.

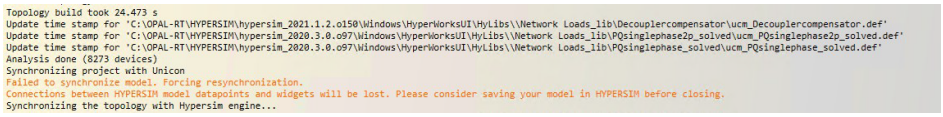


Figure 20: Screenshot of HYPERSIM warning message indicating an OPAL-RT workaround to the Unicon service failure problem.

It should be noted that the error and warning messages shown in Figure 19 and Figure 20, respectively, both occur following the “Synchronizing project with Unicon” message.

The issues encountered during the HYPERSIM version migration significantly impacted the IEEE 123-node Model develop effort. This delayed the Validation Team’s progress towards having an adequate test model for studying SLAC3R Cooperative Control. While these specific issues were resolved by the middle of March, other bugs in the HYPERSIM UI (that continue to occur intermittently at the time of this report) caused increased time required to further develop the IEEE 123-node Model, integrate the SLAC3R Cooperative Control, and conduct scenarios to test the system’s resilience improvement. The impacts to HYPERSIM activities by bugs varied from being required to close and re-open model files to requesting help from PNNL staff who

maintain the OPAL-RT systems to perform hard reboots or fix corrupted files on the simulators.

In total, 41 instances of HYPERSIM bugs impacting Validation Team work were recorded through conversations over Microsoft Teams or through individual efforts to track bugs. However, many more instances of bugs are suspected to have gone unrecorded due to the tedious and time consuming nature of bug documentation. Bug documentation was rarely presented to or discussed with OPAL-RT support because little to no resolution was expected and time spent meeting with OPAL-RT support and attempting to reproduce intermittent bugs ultimately comes at the cost of lost timeline for project deliverables.

6.2 Single-Phase Dynamic Load Modeling

The IEEE 123-node Model includes many single-phase loads. During the RD2C project's earlier stages, no single-phase load model was available in HYPERSIM. PNNL developed a user coded model (UCM) of a constant-power, single-phase load, designed to capture the loads' dependence on the system voltage magnitude better than a network of passive components. OPAL-RT recently added a built-in single-phase load model based on the UCM developed by PNNL. The Validation Team undertook to replace the UCM's with the new, built-in models to improve the efficiency of the simulation as UCMs generally incur higher OPAL-RT simulator CPU overhead than equivalent, built-in models.

An automated process to convert the UCMs to built-in models was unavailable. As a result, significant effort was spent manually replacing the UCMs. Unfortunately, upon testing the new IEEE 123-node Model featuring the built-in dynamic single-phase load models it was found that the new model was badly, numerically unstable. With the only change being the dynamic single-phase load models, the built-in model was tested independently to determine how it would respond to the severe bus voltage transients that typically occur during the beginning of the IEEE 123-node Model simulations. It was found that these models exhibited unexpected behavior during close-in faults. This behavior is presented in a comparison of load model transient behaviors provided in Section 6.2.2. This led to the following recommendations.

1. The new built-in load model should not be used for transient simulation due to producing highly unrealistic current and voltage waveforms during close-in faults.

2. Better undervoltage controls or limits should be added to the PNNL load model for improved accuracy during voltage disturbances.

Based on the recommendations, the PNNL single-phase dynamic load UCM was updated to improve the transient response to close-in faults. This process and the results are documented in the following subsections.

6.2.1 Single-Phase Dynamic Load Model Test Circuits

Figure 21 illustrates a schematic of the HYPERSIM model used to test the dynamic performance of the load models. A series RL load (R1, R2, L1) is simulated as well, providing an easy-to-analyze basis for comparison. Table 3 provides a summary of the test circuit parameters.

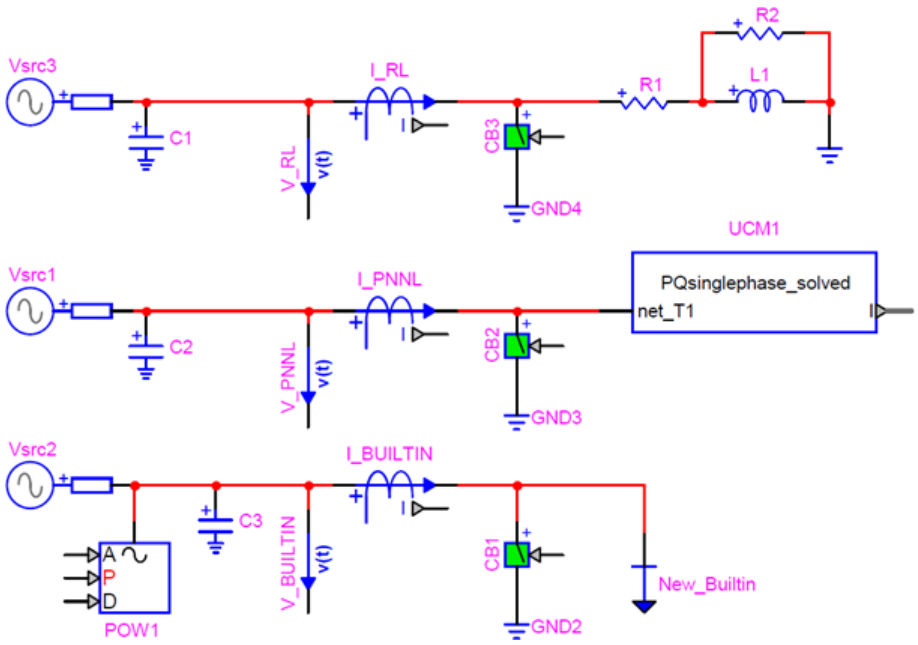


Figure 21: Schematic of HYPERSIM model used to evaluate load model dynamics.

Table 3: Network Parameters Used to Evaluate Load Model Dynamics

Parameter	Value	Parameter	Value
Source Magnitude, kV L-G RMS	10	Source Frequency, Hz	60
Source Resistance, Ω	0.1	Source Inductance, mH	5
Source Capacitance, μF	1	Passive Load, R1, Ω	80
Passive Load, R2, $k\Omega$	100	Passive Load, L1, mH	100
Constant Power Loads, MW	1	Constant Power Loads, MVAR	0.5

6.2.2 Existing Single-Phase Dynamic Load Model Test Results

The case of a close-in fault produced the findings which motivated the recommendations from Section 6.2. Fault incidence and clearing are generally a useful stress tests for dynamic models because faults are a regular occurrence in real power systems and produce significant rates of change in network currents and voltages.

Figure 22 illustrates the line current and voltage (refer to the current sensors shown in Figure 21) for a short-circuit event. The short-circuit occurs at t = 0.1 s and is cleared at t = 0.15 s. From the perspective of load dynamics, the period following fault clearing is of the most interest. A subset of this time period is shown in Figure 23 to make the transients of importance easier to see.

The passive load exhibits a behavior commonly seen in transient studies. When the fault

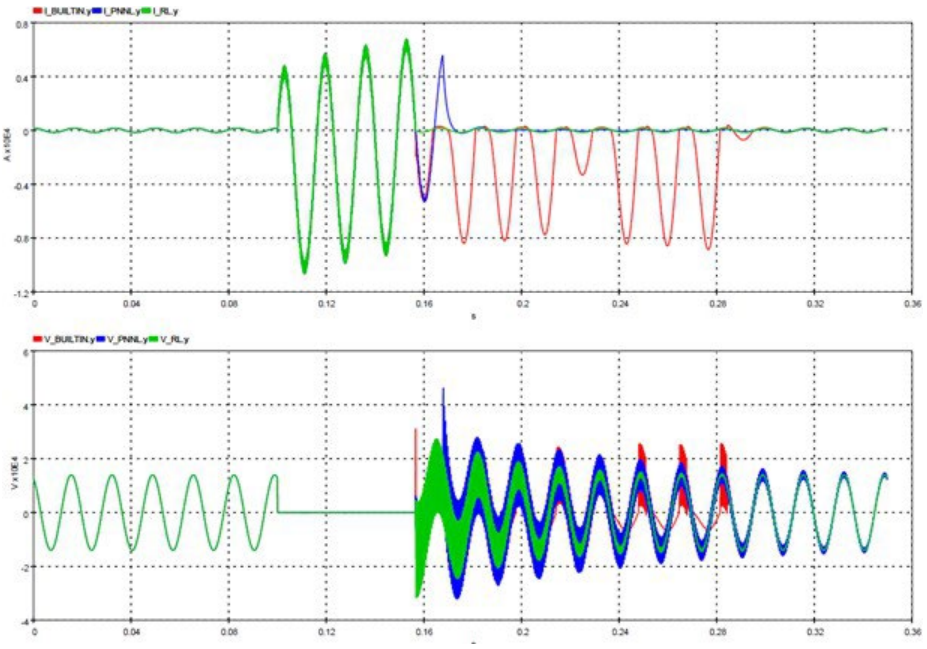


Figure 22: Response of existing load models to incidence and clearing of a short circuit event.

clears, a transient recovery voltage (TRV) occurs, having a high frequency due to the relatively small shunt capacitance and inductance of the source. The load current returns to its nominal value nearly immediately after the fault clears.

The PNNL-designed load model exhibits some mildly unrealistic behavior that could affect RD2C model accuracy in specific circumstances. After the fault clears, the load consumes significant current for roughly 10 ms. This consumption is likely due to the wind-up of the load model's control that occurs during the fault. While some load pickup can occur following a fault or outage, the observed value is multiple orders of magnitude greater than observed in practice.

The built-in model provided by HYPERSIM exhibits highly unrealistic behavior that is likely to cause major errors or even instabilities in the RD2C model. Following the fault clearing, the built-in model consumes a highly asymmetric current with intermittently varying peak values. This highly nonsinusoidal current is likely to cause unusual operation in any nearby inverters or saturable inductances. The asymmetric current waveform also causes a highly unusual TRV pattern, which is likely to produce unrealistic waveforms in voltages throughout the network.

6.2.3 Improved PNNL Single-Phase Load Model Test Results

The PNNL load model is internally implemented as a variable resistance and inductance. This type of implementation allows for a simple solution to the unwanted post-fault overcurrents observed previously in Figure 22.

In the existing implementation, an RMS voltage measurement is used to update the resistance and inductance needed to maintain the desired real and reactive power. In the modified model, this update process is carried out only if the RMS voltage exceeds a user-set threshold. This

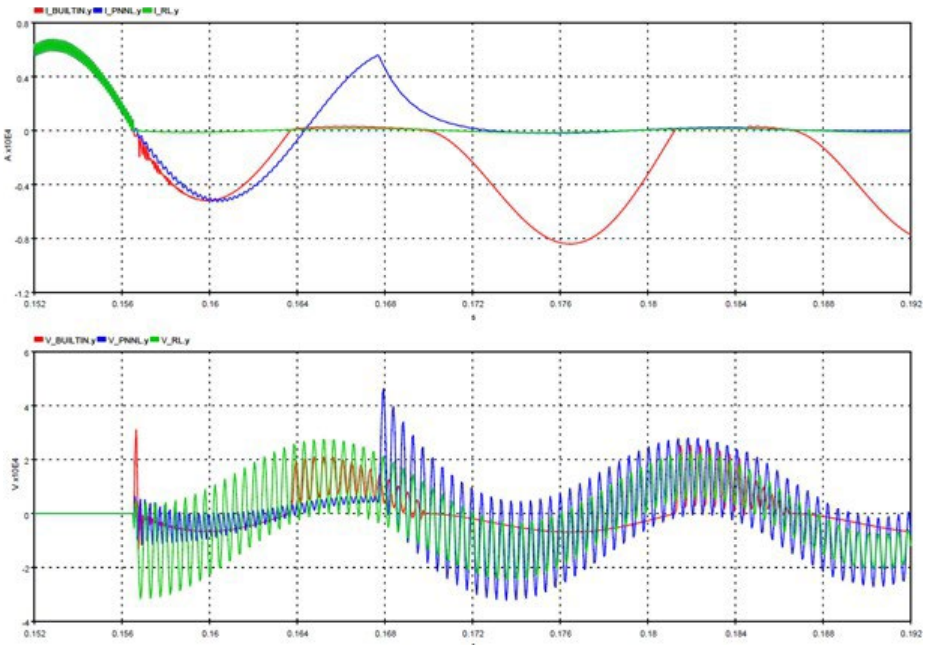


Figure 23: Magnified view of Figure 22, cropped to illustrate load dynamics following fault clearing.

approach is used in several commercial EMT programs; it allows the dynamic load model to respond to minor disturbances such as voltage regulation and voltage sags or swells, which the load model is intended to respond to. However, it prevents unrealistic overcurrents such as those seen in the existing load model.

Figure 24 illustrates the short-circuit response of the three dynamic load models. The PNNL model results shown reflect the improved undervoltage control. The one-cycle overcurrent observed previously in the PNNL model is now absent. Once the fault clears, the system RMS voltage is briefly increased as a result of the TRV. This causes the temporary reduction in load current seen in the PNNL model, which is in accordance with the constant power characteristic that the load model is intended to implement.

In conclusion, significant effort by the Validation Team was performed to study the behavior of the built-in single-phase dynamic loads models, propose an alternative, and implement changes to the existing PNNL UCM. This effort was unavoidable because the poor transient performance of the built-in load models could not have reasonably been expected from an EMT type simulation platform and the resulting numerical instability of the new IEEE 123-node Model forced a reversal of the significant, initial effort to replace the PNNL UCMs. This effort would have been avoided had the OPAL-RT built-in single-phase dynamic load models been properly coded to exhibit acceptable, transient behavior in EMT type simulations.

6.3 Performance and Limitations of HYPERSIM Python API and ScopeView

The Python API for HYPERSIM has been selected as a promising tool to automate the execution of tests for SLAC3R cooperative control. However, the existing documentation for the

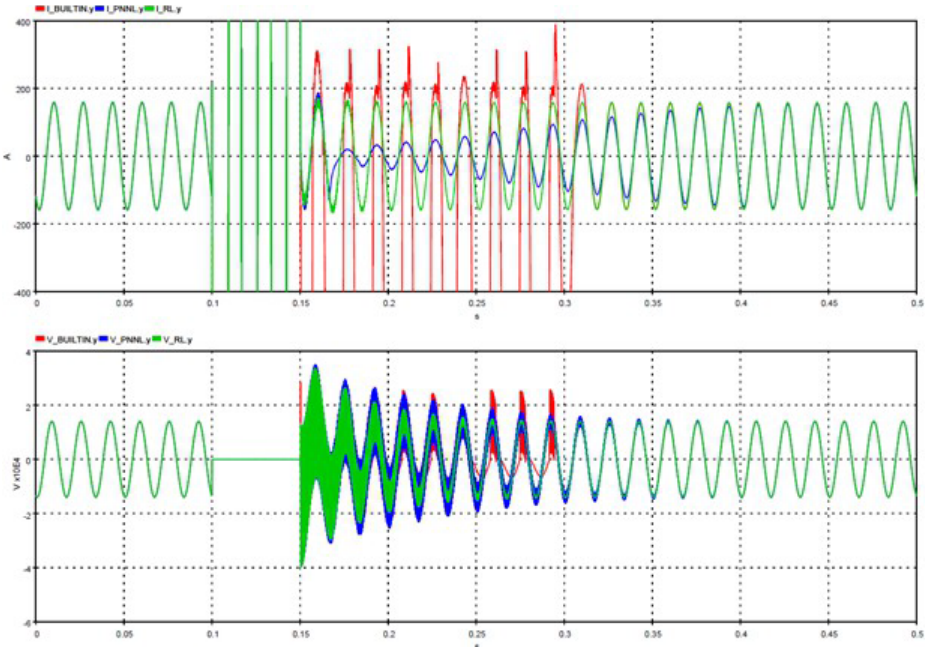


Figure 24: Short-circuit response of dynamic load models, illustrating improvments to PNNL load model's post-fault response.

API is sparse, and some testing was conducted to better understand the API's performance and limitations. ScopeView has been used on an ongoing basis for simulation monitoring. During the course of API testing, it was discovered that ScopeView has some serious limitations (possibly some of which are unintended). Thus, the scope of testing was expanded to include some further testing of ScopeView as well.

Initially, it was hoped that the API could provide some control while the real-time simulation was running. This control would be used to either emulate slow-acting controls (e.g., at a one or two second update rate) or to simulate complex scenarios that would be cumbersome to implement completely in the HYPERSIM controls. However, it has been determined that the API cannot do this without compromising simulation accuracy. Furthermore, it has been determined that even retrieving plot data via ScopeView while the simulation is running compromises test accuracy. This testing resulted in the establishment of two new design practices:

1. If possible, the Python API should not be called when the simulation is running. If using the API during runtime is necessary, a detailed assessment of the API's effects on simulation execution time should be performed. The simulation timestep will likely need to be at least 125 µs.

2. ScopeView can be used for spot-checks and troubleshooting, but should not be used for formal analysis. It is possible that ScopeView will not cause issues in simpler models or at much larger timesteps (e.g., 500 µs). However, exploring the boundaries of the identified issues is beyond the scope of this testing.

For SLAC3R testing, the role of the Python API will be restricted solely to automating offline model configuration and starting and stopping the simulation. The datalogger will be used in the

place of ScopeView for formal analyses.

6.3.1 Findings of the HYPERSIM Python API and ScopeView Investigation

All of the reported results were obtained by running the model on Opal-RT hardware.

- While ScopeView is suitable for troubleshooting and quick assessments, it should be used with extreme caution for formal analysis. This is because the process of triggering a Scope-View capture upsets the simulation, and this has been observed to have significant effects on simulation accuracy.
- The API is unsuitable for any kind of closed loop control of a running HYPERSIM because the API accesses themselves can significantly degrade the accuracy of the simulation results.
- The API should not be called while a simulation is running if possible. If this is necessary, it should not coincide with any simulation events. Based on the transients that could be captured in Scopeview, several seconds of simulation time should pass between API calls and the start of any captures that are to be used for analysis.

These recommendations are based on the determination that the API is subject to the following limitations/restrictions:

- The API communications path is shared with ScopeView. In other words, if Scopeview is currently retrieving data from the target, you cannot send or receive any information via the API.
- API calls, regardless of their frequency, appear to generate stretched steps (i.e., non-realtime operation).
  - This was verified using a model running on a 50 µs timestep. Processor burden without API calls was verified by running a long simulation (e.g., several minutes of simulation time)
- Scopeview calls are subject to the same issue. This appears to visibly affect simulation results—meaning that the Scopeview call itself creates a transient in the model, even if there are no programmed events occurring.
  - In some cases, ScopeView plot updates produced inaccurate model results even when they did not produce stretched timesteps. The fact that a HYPERSIM model does not report stretched timesteps does not necessarily mean that ScopeView plot updates have not corrupted results.
- For reasons that could not be determined, updating the gain of the constant block in HYPER-SIM via the Python API does not always successfully update the gain within the model that is actually running on the target.

6.3.2 HYPERSIM Python API Test Methodology

The initial goal of the API testing was to characterize the latency between the API and a simulation running in real time on Opal-RT hardware. To this end, a simple loopback test was designed, intended to represent a simple control that would represent best-case conditions for API performance. Prior to this testing, basic aspects of API performance such as reliability and the handling of multiple API requests were unknown, so the test was intended to be a simple proof-of-concept.

The test signal for the loopback test was a sinusoidal signal source placed in HYPERSIM (API_OUT). This signal was read into an instance of Windows command line using the Python API along with the simulation timestep. The signal was then fed back into HYPERSIM via the API by changing the value of a constant in the model (API_IN). Figure 25 illustrates the logical flow of the test signals and the measurement points available.

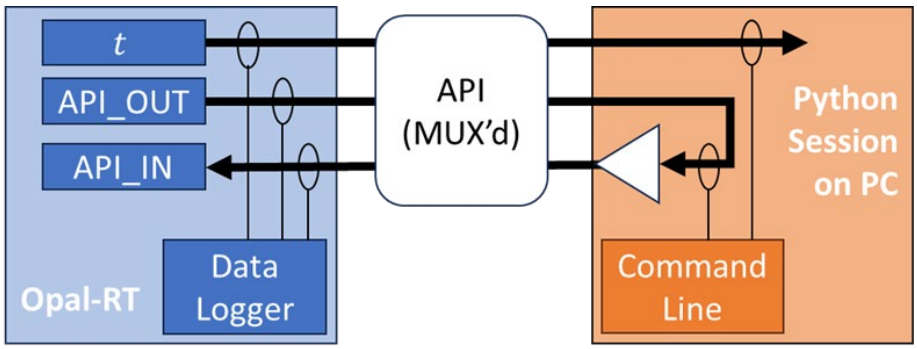


Figure 25: Signals used in loopback testing and their logical flow.

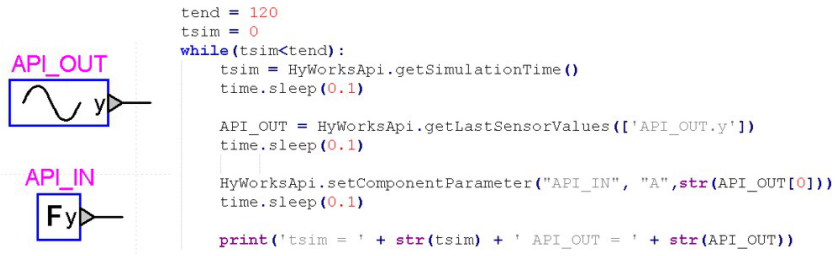
The signals that flow through the Python API are multiplexed (MUX'd). The API serves calls sequentially, and waits for each call to complete before proceeding to the next. A single call can only be used to perform relatively simple commands, such as changing or reading a single simulation value. For example, the API could not collect both simulation time and the API_OUT signal in the same call. This leads to an unavoidable time de-synchronization between any data that are collected. One goal for this testing was to measure how quickly the API could serve calls, which would determine the severity of the time de-synchronization and the precision of the measurement and control that could be achieved.

Figure 26 illustrates the HYPERSIM components (left) and the portion of the Python API code used for the loopback test (right).

The use of deliberate time delays (calls of time.sleep) was intended to reduce the risk of API or simulation malfunctions by throttling the rate at which API requests were made to less than ten times a second.

6.3.3 HYPERSIM Python API Test Results

All simulations were performed on Opal-RT hardware running with a simulation timestep of 50 µs.




```
tend = 120
tsim = 0
while(tsim<tend):
    tsim = HyWorksApi.getSimulationTime()
    time.sleep(0.1)

    API_OUT = HyWorksApi.getLastSensorValues(['API_OUT.y'])
    time.sleep(0.1)

    HyWorksApi.setComponentParameter("API_IN", "A",str(API_OUT[0]))
    time.sleep(0.1)

    print('tsim = ' + str(tsim) + ' API_OUT = ' + str(API_OUT))
```

Figure 26: HYPERSIM model components (left) and Python API code (rIght) for loopback testing.

Loopback Testing

The Python API successfully read the simulation time and value of API_OUT from the Opal-RT hardware. The mean time to serve an API call was approximately 150 ms with a standard deviation of 50 ms. Based on this observed time delay, the frequency of API_OUT was set to 0.05 Hz (a period of 20 s). Figure 27 illustrates a sample of the API_OUT data collected. The orange trace illustrates the values of API_OUT plotted against the retrieved simulation time. The blue trace illustrates the value of API_OUT as computed using Equation 1:

$$API_{OUT} = sin(2\pi \cdot 0.05 \cdot t_{sim}) \quad (2)$$

where t_sim is the simulation time reported by the API. For this testing, calls to time.sleep between collecting the simulation time and API_OUT were disabled.

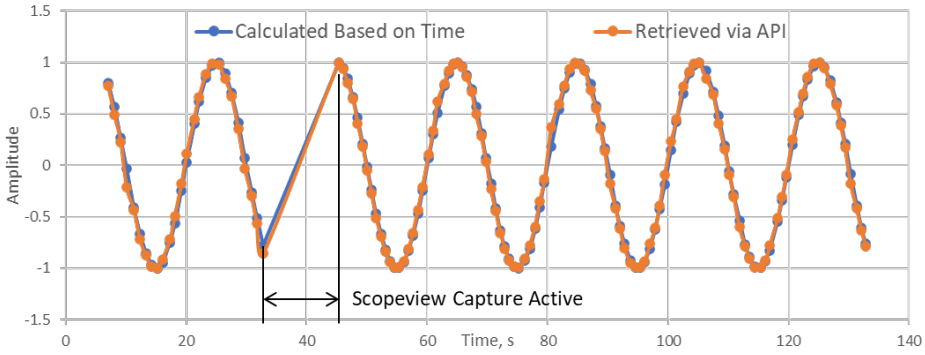


Figure 27: Measuring of API_OUT and API_IN Via HYPERSIM Datalogger.

The difference between the blue and orange traces is due to the time de-synchronization discussed earlier—some simulation time passes between the moment where the simulation time is collected and the value of API_OUT is collected. This limits the minimum timescale on which accurate control can be performed to roughly several seconds.

Observe that data is absent for times in the range of roughly 26 to 44 seconds. A ScopeView capture was manually triggered (i.e., the API was not used to control ScopeView) at this time period. This indicates that the communications channel used for serving API calls is also used for collecting ScopeView data. Thus, ScopeView cannot be used contemporaneously with any

active simulation control involving the API.

The value of API_OUT collected via the API was successfully transmitted back into the Opal-RT simulation, albeit with intermittent success. Figure 28 illustrates data collected from the Opal-RT hardware using the data logger. The blue trace illustrates API_OUT. The red trace illustrates the output value of API_IN.

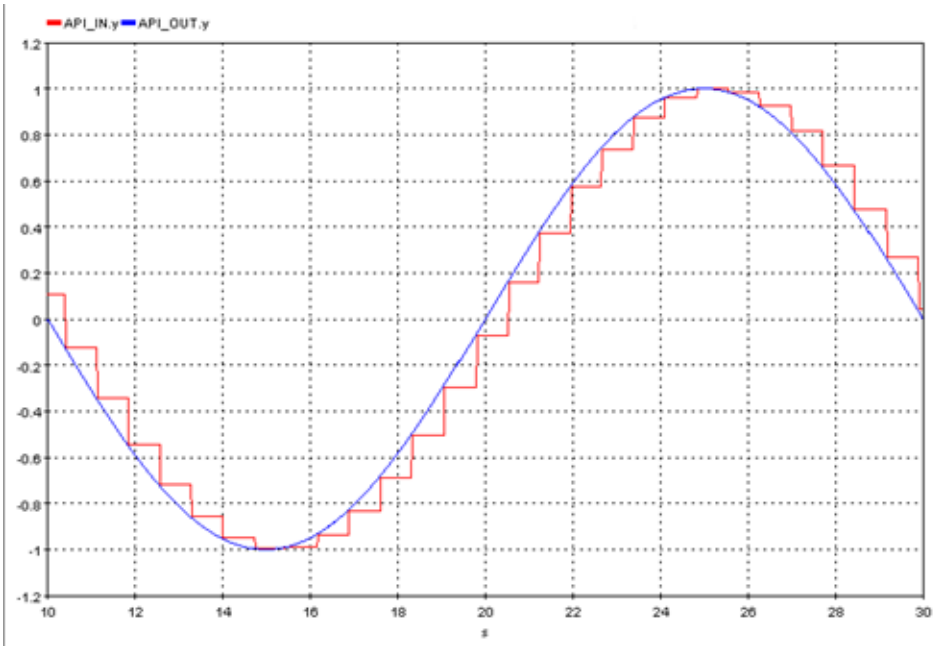


Figure 28: Sampling of API_OUT via Windows command line using Python API.

API_IN is a constant in the HYPERSIM model, and so the fact that its output values are changing verifies that the Opal-RT hardware is receiving data from the Python API and reflecting it in the model. The phase shift between the two sine waves represents the total round-trip delay of the signal path shown in Figure 25.

Upon being re-opened for further testing at a later date, this same model was unable to reproduce the behavior shown in Figure 28. Instead, changes to the constant value could be observed in the HYPERSIM application but did not appear to be reflected on the Opal-RT hardware. This suggests that some undocumented or unintended software behavior is affecting the ability of changes in constant blocks to be reflected on the target while it is running. This issue may be associated with the data recorder—instances were observed where changes to data recorder settings did not appear to be reflected in data recorder behavior until HYPERSIM had been restarted and model generation tasks repeated several times.

Further troubleshooting of the API loopback testing is unlikely to be useful—more serious limitations involving stretched timesteps have been discovered that seriously limit the usefulness of the Python API and Scopeview.

Stretched Timesteps During Plotting or API Calls

It was observed that, when the Python API was used, HYPERSIM reported stretched timesteps. The presence of stretched timesteps indicates that the Opal-RT hardware was unable to complete its computations within the specified timestep (50 μs), and that the results may be inaccurate. As an initial diagnostic test, an extended simulation was run and the Real-Time Monitor was used to track execution and communications time. The real time monitor results for the initial diagnostic test is presented in tabular form in Table 4.

Table 4: Real-Time Monitoring Results for Baseline Test (No ScopeView or API Calls

Processor	Maximum Execution Time, μs	Maximum Communications Time μs	Total Time	Remaining Margin
1	37.48	0.24	37.72	25%
2	25.13	0.15	25.28	49%
3	20.95	0.14	21.09	58%

Because it was observed during prior testing that ScopeView data captures occupy the same communications path as Python API calls, it was hypothesized that ScopeView might cause similar issues related to stretched timesteps. This hypothesis was also motivated by the fact that ScopeView captures sometimes appeared to contain transients at their time of initiation that were not obviously linked to any event within the model (Figure 29):

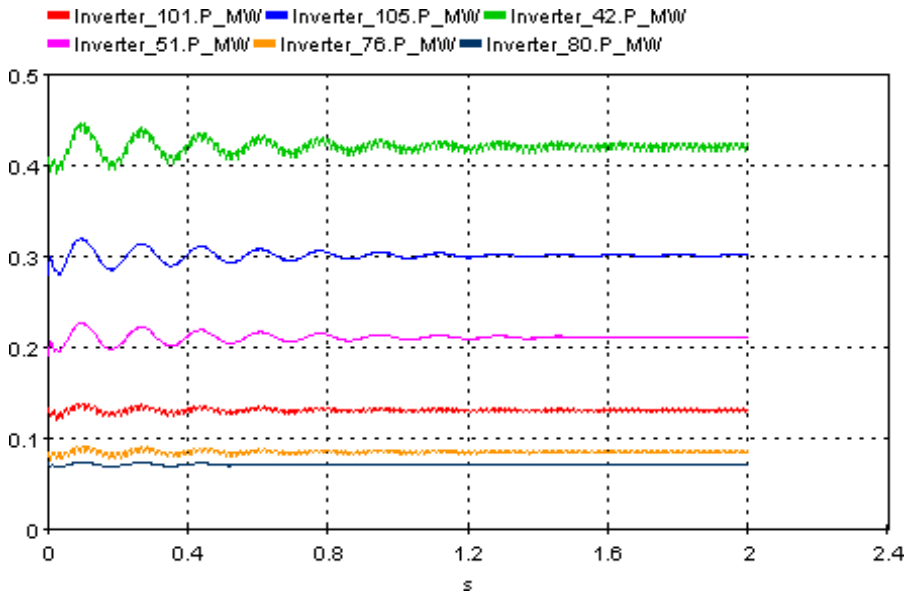


Figure 29: A transient in inverter powers that is present at the start of a ScopeView capture. There is no step-change occurring in the model at t=0. The signals in this plot should exhibit steady state behavior.

Given that this kind of behavior is easily introduced by user error, fairly detailed testing was

required to ascertain whether this is due to an unintended behavior endemic to ScopeView or HYPERSIM. To further test this hypothesis, the Datalogger was set to record continuously. ScopeView and the Python API were applied to a long simulation and the effects on inverter outputs observed. As a control, a long steady-state simulation was run (Figure 30). Regardless of whether the Python API is called or Scopeview is triggered, the resulting inverter outputs should match Figure 30 exactly.

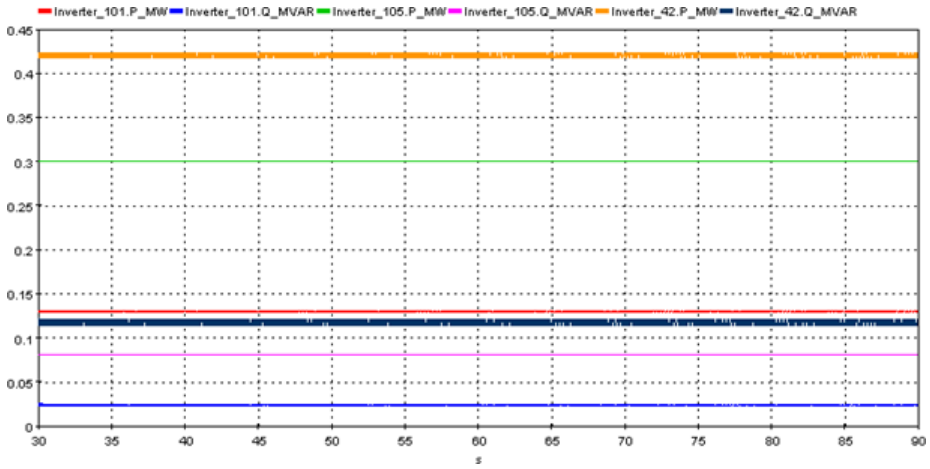


Figure 30: Steady-state inverter powers measured in control simulation — subsequent tests should yield the same output.

Effects of ScopeView Calls

To test the effects of Scopeview, the simulation shown in Figure 30 was repeated but Scopeview captures were manually taken at 35, 45.7, and 74.8 s. Figure 31 illustrates the resulting inverter real and reactive power outputs. The “Sync” and “Trigger” options in ScopeView were left unselected, and all programmable simulation events were disabled—all due diligence has been exercised to ensure no other disturbances could have been applied to the simulation at these times.

The results shown in Figure 31 suggest that the act of initiating a ScopeView capture causes spurious transients in the model. Analysis with the HYPERSIM Real-Time Monitor suggests that this error is primarily caused by stretched timesteps—collecting a ScopeView capture appeared to increase simulation execution time by more than 75 µs, making it impossible to meet the real-time deadline of 50 µs.

It was hypothesized that the severity of the stretched timestep issue could be reduced by either capturing fewer signals or by decreasing the simulation execution time. To test this, the Real-Time Monitor was used to collect worst-case execution time for both the full signal set (94 signals) and a single signal. A capture length of 1 second and a decimation factor of 10 (giving a sampling frequency of 1 kHz) was used. For the SLAC3R testing, this represents generally the lower limit of what would be a useful quantity of data to collect.

Testing consisted of collecting 10 to 20 manually-triggered ScopeView captures and observing how the captures affected simulation execution time and the quantity of stretched timesteps.

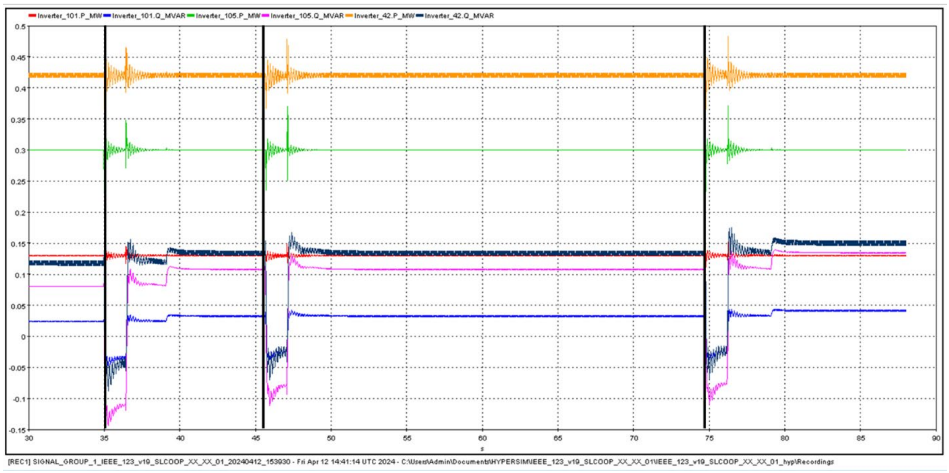


Figure 31: Unexpected transients in power converter outputs. Vertical thick black lines indicate times when ScopeView captures were taken.

Table 5 illustrates Real-Time Monitor results for the full signal set.

Table 5: Real-Time Monitoring Results for ScopeView Calls, Measuring Full SLAC3R Signal Set

Processor	Maximum Execution Time, μs	Maximum Communications Time μs	Total Time	Remaining Margin
1	167.17	0.15	167.32	-235%
2	109.98	0.33	110.31	-121%
3	84.17	0.19	84.36	-69%

The results in Table 5, in concert with those in Figure 31, confirm the initial hypothesis that the ScopeView captures are causing stretched timesteps, and that the resulting overage is far in excess of what the Opal-RT's processors can handle. Each capture triggered was observed to produce five or six missed timesteps.

Table 6 illustrates Real-Time Monitor results for the capture of a single signal (API_OUT). This capture is a small amount of data (2000 points). It is very reasonable to expect that it will not excessively burden the Opal-RT processors to collect and distribute this capture.

The values in Table 6 indicate that, even when collecting only a relatively small number of samples (2000) for a single signal, a delay of more than 50 µs is introduced (see Table 4 for a comparison). Two to three stretched timesteps were created for each capture. Moreover, stretched steps were not confined to the processor on which the associated signal was present (processor 1), but all three processors missed timesteps.

The extra execution time introduced by sampling API_OUT exceeds 50 µs. Thus, even if there was no time required to solve the rest of the model, stretched timesteps would still be present.

Table 6: Real-Time Monitoring Results for ScopeView Calls, Measuring a Single Signal

Processor	Maximum Execution Time, μs	Maximum Communications Time μs	Total Time	Remaining Margin
1	111.74	0.20	111.94	-124%
2	76.67	0.19	76.86	-54%
3	61.03	0.59	61.62	-23%

Therefore, ScopeView appears unsuitable for use in models having a 50 µs timestep.

The additional execution time (t_{exec}) introduced by calling ScopeView can be roughly estimated according to Equation 3:

$$t_{exec} = 75 + \frac{N_{dp}}{3400}\mu s \tag{3}$$

where N_{dp} is the number of datapoints being collected.

This is a very rough estimate, as the actual time likely depends on the model under study and how the tasks are mapped to the different processors. Unless there are components in the SLAC3R control model that make ScopeView calls take far longer than usual (this would be an undocumented and unexpected behavior), ScopeView is unlikely to be useful for simulations with timesteps on the order of tens or sometimes even hundreds of microseconds.

Sporadically throughout testing, it was observed that ScopeView captures appeared to cause model transients even when no stretched timesteps were present. This was reproduced in a more consistent format by capturing a small amount of data (100 points from one signal, capture length of 50 ms, synch on). Seven captures were manually triggered at random times. Figure 32 illustrates inverter powers for the duration of the test. The first three captures caused stretched timesteps. The last four, however, did not.

Effects of API Calls

To test the effects of the Python API on stretched timesteps, an experiment was conducted with the loopback test described in previously in Figure 25. A long simulation and was run and the API was set to perform queries at a rate of 10 Hz. Figure 33 illustrates the same inverter powers shown back in Figure 30. Theoretically, these two plots should be identical.

The results show significant distortion: every API call appears to disturb the simulation in a manner similar to triggering a ScopeView capture. Thus, calling the Python API during a running simulation appears to render the simulation results completely invalid.

Based on the available data, it is believed that the transients shown in Figure 32 and Figure 33 are caused by stretched timesteps, and that these stretched timesteps are caused either by API calls or ScopeView captures. The effect of the stretched timesteps on simulation accuracy is clearly significant, particularly on the reactive power outputs of the inverters.

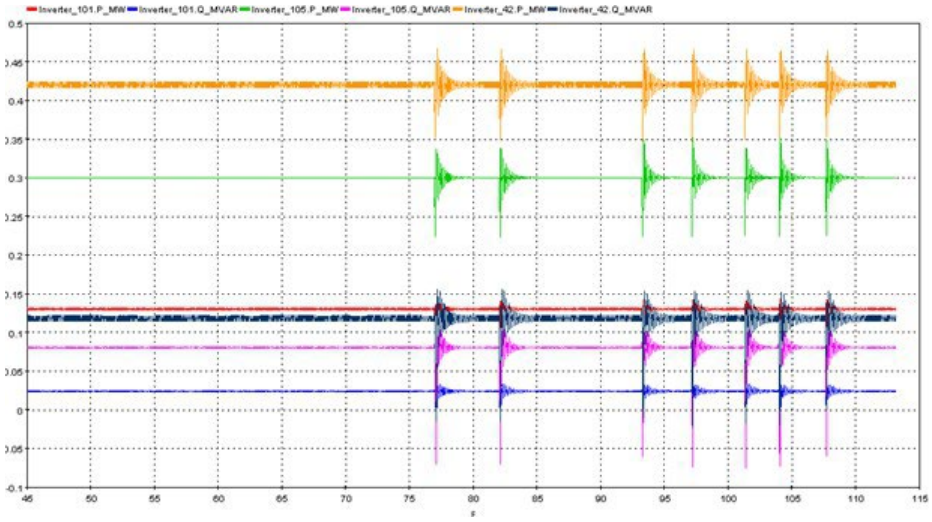


Figure 32: Datalogger capture of steady state simulation. Transients indicate moments in time where ScopeView captures were taken.

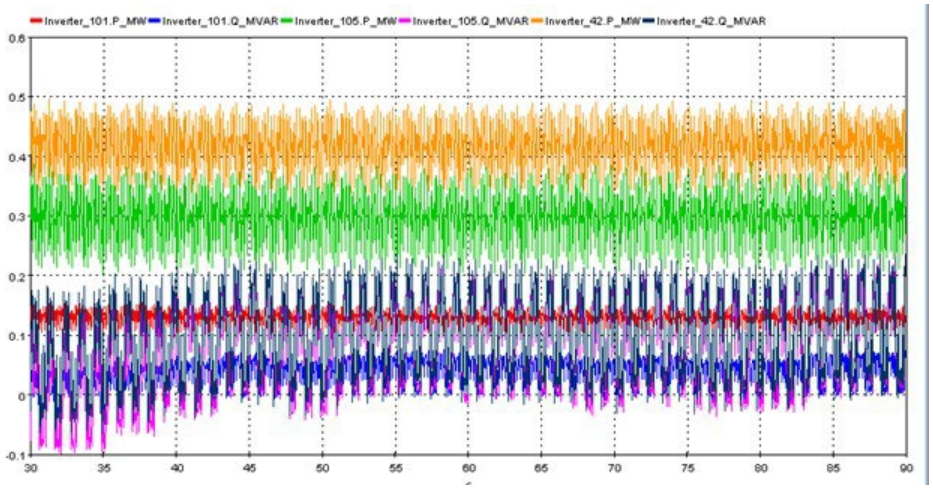


Figure 33: Inverter powers for steady-state simulation with repeated Python API calls. All values should be relatively constant.

Table 7 illustrates Opal-RT performance as measured, using the Real-Time Monitor, for the simulation when API calls, throttled to an interval of 100 ms or more, are performed.

These results indicate that, as in the case of ScopeView, calling the Python API introduces significant processor overhead (roughly 30 µs in this case). However, the time required to serve the API is much shorter than that required to serve even a small ScopeView capture. These results suggest that it is technically possible (although likely impractical) to support some API functionality while keeping a simulation timestep of 50 µs.

Given the results shown in Figure 33, it appears that even relatively small time overruns cause significant simulation errors. While the maximum overrun was 28%, many of the execution

Table 7: Real-Time Monitoring Performance for API Calls Throttled to 100 ms Interval

Processor	Maximum Execution Time, μs	Maximum Communications Time μs	Total Time	Remaining Margin
1	63.55	0.23	63.78	-28%
2	42.14	0.19	42.33	15%
3	40.52	0.24	40.76	18%

times for API calls were much less severe (e.g., <5%). Thus, whatever method Opal-RT uses to handle stretched timesteps (e.g., using the previous timestep value) is sensitive to even small stretches.

Removing the sleep functions (and thus the throttling) was observed to result in larger worst-case execution times (Table 8).

Table 8: Real-Time Monitoring Performance for Unthrottled API Calls

Processor	Maximum Execution Time, μs	Maximum Communications Time μs	Total Time	Remaining Margin
1	109.55	0.24	109.79	-120%
2	73.50	0.18	73.68	-47%
3	49.78	0.22	50.00	0%

Most of the observed execution times were similar to those without the added sleeps. However, the maximum overrun was significantly greater. This suggests that failing to rate limit the API queries does in fact cause occasional congestion and increased execution time. However, due to the high variance observed in execution time, this cannot be said definitively. It is possible that execution times greater than 100 µs could occur even in the presence of throttling but are simply too rare to have been observed during testing.

While there remain further questions as to the limits of API performance and how the model might be adapted to allow for online simulation, the analysis done so far is sufficient to define some constraints that will allow project goals to be met while mitigating the limitations of HYPERSIM, ScopeView, and the Python API. These constraints were listed previously in Section 6.3.1 and their relevance to RD2C is summarized in Section 6.3.

Although the time spent to investigate the HYPERSIM Python API and ScopeView impact on real-time simulation fidelity was substantial, the lessons learned and intuition gained about the OPAL-RT HYPERSIM platformed proved valuable. The Validation Team discovered that using the Python API to instantiate simulation events on the fly would compromise simulation fidelity and jeopardize the resilience assessments of controls implemented in external hardware (e.g. SLAC3R Cooperative Control running on Cybernet VMs). Additionally, the team discovered that

in order to collect data from a real-time test of the SLAC3R Cooperative Control, the Datalogger tool would need to be used. The team was able to act on this knowledge to implement a plan to trigger simulation events through model logic that can be configured offline prior to simulation start, thus avoiding use of the Python API during the simulation. Additionally, a new data capture plan was implemented to make use of the Datalogger tool and integrate the tool settings into the HYPERSIM Python automation scripting. While the team was ultimately successful in mitigating the impacts discussed in this section, implementing these strategies increased the model development and simulation automation scripting effort.

7.0 Conclusions and next steps

The main goal of the FY24 has been integration of more resilience-driven mitigation strategies within the closer-to-real-life simulation platform driven by Opal-RT HYPERSIM capabilities. In the previous year, the SLAC3R local safety controller has been incorporated in an IEEE 123-node system modeled in HYPERSIM and including 3 GFLs, and 3 GFMs with SLAC3R. For the task of enhancing the SLAC3R controllers to include their cooperative algorithm, the 3 GFLs have been replaced by GFMs. In the process, due to the fact that the base case results turned out to be unstable and unrealistic, the HYPERSIM inverter and load models had to be re-tested and improved. This document presented many details on these challenges and the solutions the team adopted to resolve them, such that further investigations and scenario developments and tests will not be hindered.

Moreover, to expedite different scenarios simulation and data gathering, a set of automation scripts have been developed. These scripts rely on the HYPERSIM Python APIs to configure scenarios and use-cases, run them, and acquire and save data into formats that could then be post-processed.

The new developments in terms of system modeling and scenario simulations have been applied to gather data on two specific scenarios, which have then been analyzed from resilience perspective. In these particular studies, the goal has been to find some metrics that would quantitatively measure how the mitigation strategies helped maintaining or improving the system resilience related to two indexes: ability to provide frequency support to the microgrids, and ability to alleviate stress on slower synchronous generators in the race to sustain critical load. A series of performance-based metrics have been calculated, applied on the system frequency, and tentatively visualized as a radar graph to conclude on system resilience. Given the EAC comments and our current investigations, we believe that are still some work to be done in order to come with a better way of interpreting the numbers and the graphs.

As the end of the FY24 brought a twist in the project life cycle, the work of this project is going to be continued under the RD2C's capstone project entitled 'Integration and Validation of Multi-Layer Mitigation Strategies for Cyber Physical Systems Resilience'. This project will continue incorporating and testing resilience-driven mitigation strategies at larger scale on high-fidelity system models with a high penetration of inverter-based resources subjected to the consequences of unpredicted natural events or cyber-physical malicious attacks.

References

[1] M.H. Amirioun et al. “Metrics and quantitative framework for assessing microgrid resilience against windstorms”. In: *International Journal of Electrical Power & Energy Systems* 104 (2019), pp. 716–723. ISSN: 0142-0615. DOI: https://doi.org/10.1016/j.ijepes.2018.07.025. URL: https://www.sciencedirect.com/science/article/pii/S0142061518300097.

Pacific Northwest National Laboratory

902 Battelle Boulevard
P.O. Box 999
Richland, WA 99352
1-888-375-PNNL (7675)

www.pnnl.gov