Implementation and Validation of PLC Emulation and Data Transfer

Mohamed El-Genk[1,2], Timothy Schriener[1,2],

Christopher Lamb[3], Raymond Fasano[1,2,3], Andrew Hahn[1,2]

[1]Institute for Space and Nuclear Power Studies, University of New Mexico, Albuquerque, NM, USA

[2]Nuclear Engineering Department, University of New Mexico, Albuquerque, NM, USA

[3]Sandia National Laboratories, Albuquerque, NM, USA

Technical Work Scope

Nuclear Energy–Cybersecurity Research Topics and Metrics Analysis (NE-1)

DOE-NEUP Project 18-15055.

DOE Contract No. Nu-18-NM-UNM-050101-01 to the University of New Mexico.

Period of Performance: 07-01-2018 to 06-30-2021

Progress Report No. UNM-ISNPS-02-2019

Institute for Space and Nuclear Power Studies, the University of New Mexico

Albuquerque, NM, USA

July 2019

Executive Summary

The use of digital instrumentation and control (I&C) systems within commercial nuclear power plants raises cybersecurity concern in the nuclear industry, with an increasing need for high fidelity cybersecurity analyses of the I&C architectures within nuclear power plants. To meet this need, the Nuclear Instrumentation & Control Simulation (NICSim) platform is currently being developed at the University of New Mexico's Institute for Space and Nuclear Power Studies (UNM-ISNPS) in collaboration with Sandia National Laboratories (SNL), under a DOE NEUP award. It aims to develop a platform that couples emulation and simulation models of the digital I&C system components to a dynamic model of a PWR power plant. This capability would enable cybersecurity investigations of the safety I&C systems of commercial nuclear power plants.

The development of the NICSim platform requires developing a validated programmable logic controllers (PLC) emulation methodology for modeling different controllers within the nuclear power plant's I&C architecture. Thus, a communication interface is needed to link the emulated PLCs to a physics-based model of the nuclear plant, which is the primary focus of this project. The work presented in this progress report helps address this need by developing and validating a PLC emulation methodology and developing a reliable, fast-running interface that effectively links a PLC to a physics-based simulation model within Matlab Simulink framework.

The developed emulation methodology characterizes the PLC's digital and physical signatures. It also validates the emulated PLC that its behavior matches that of the real hardware. The validation results show the actuation responses, sampling rates, and network responses of the emulated PLC are indistinguishable from those of the real PLC. From a cybersecurity perspective, the PLC emulation runs the same operating system and software, uses the same communication protocols, and generates the same types and proportions of network traffic data as the real device. The developed data transfer interface links a PLC and the Simulink model, transmitting the state variables to the PLC and transmitting back the generated control signals to actuate system within the simulation. Testing different communication methods reveals that the shared memory method performs the best. It is fast, reliable, and synchronous. The effectiveness of this interface is successfully demonstrated for linking a PLC to a Matlab Simulink, physics-based, dynamic model of an integrated space nuclear reactor power system. The results are indistinguishable from those generated using internal Simulink control logic.

List of Contents

List of Figures

List of Tables

Abbreviations List

ALIP:	Annular Linear Induction Pump
API `	Application Programming Interface
CBC:	Closed Brayton Cycle
CPU:	Central Possessing Unit
DOE:	Department of Energy
DHCP:	Dynamic Host Configuration Protocol
HITL:	Hardware in the Loop
ICS:	Industrial Control System
I&C:	Instrumentation and Control
IP:	Internet Protocol
NICSim:	Nuclear Instrumentation and Control Simulation for Modeling Cyber-Attacks
PC:	Personal Computer
PCAP:	Packet Capture
PLC:	Programmable Logic Controller
PMA:	Permanent Magnet Alternator
PWR:	Pressurized Water Reactor
S^4:	Submersion Subcritical, Safe, Space
SNL:	Sandia National Laboratories
TCP:	Transmission Control Protocol
TCP/IP:	Transmission Control Protocol over Internet Protocol
UNM-ISNPS:	University of New Mexico's Institute for Space and Nuclear Power Studies
VM:	Virtual Machine

1. Introduction

In this and past decades targeted cyber-attacks on critical energy infrastructure have occurred on Industrial Control System (ICS) networks, including prominent attack campaigns such as Stuxnet, Havex, BlackEnergy III, and CrashOverride (Dragos Inc. 2017; Karnouskos 2011). The consequences of these attacks have shown that malicious cyber-attacks can be deployed and developed to destabilize and disable specialized industrial computing equipment. These digital control systems rely on specialized computers, such as Programmable Logic Controllers (PLCs), for monitoring and autonomous control of key processes. Unlike enterprise IT networks, ICS networks frequently do not have the same cybersecurity safeguards and defensive technologies that have been developed to confront increasingly sophisticated cyber-attacks in the field of enterprise IT.

The increasing use of digital instrumentation and control (I&C) systems within commercial nuclear power plants makes the cybersecurity a major concern in the nuclear industry (National Research Council 1997; Korsah, et al. 2008). A targeted cyber-attack against a nuclear power plant could potentially have severe consequences to operation and safety. The recent infiltration of Wolf Creek's business network in 2018, a Westinghouse PWR in Kansas, demonstrates the ever present and evolving cyber threat to nuclear power plants in the United States (Perlroth 2019). As future plants are being designed with mostly or all digital I&C infrastructure there is an increasing need for high fidelity cybersecurity measures and detailed analyses of I&C architectures for nuclear power plants.

The Nuclear Instrumentation & Control Simulation (NICSim) platform currently being developed at the University of New Mexico's Institute for Space and Nuclear Power Studies (UNM-ISNPS) in collaboration with Sandia National Laboratories (SNL), under a DOE NEUP award, aims to address the current needs. It aims to develop emulytics capabilities and couple emulation and simulation models of digital I&C system components to a dynamic model of a nuclear power plant to enable cybersecurity investigations of the I&C systems. This platform will be implemented into the Department of Energy's (DOE's) SCEPTRE emulation framework (Sandia National Laboratories 2016), first developed to deal with cyber-attacks on energy grids, to emulate and simulate I&C system architectures in nuclear power plants. The NICSim platform will include models of the digital PLCs which control many of the autonomous control and safety system actuation processes within modern nuclear power plants. The emulated and

simulated I&C components will be linked to a physics-based dynamic model of a commercial Pressurized Water Reactor (PWR) nuclear power plant for direct feedback of its behavior and response of the integrated I&C systems. This dynamic model will include representative submodels of different sensors in the plant to simulate the measurement signal responses of the instruments to the PLCs.

1.1. Objectives

The development of the NICSim platform requires validated emulation of representative PLCs for use in modeling different autonomous operation and safety controllers within the plant's I&C architecture. Additionally, implementing the emulated PLCs in conjunction with the nuclear plant simulation model requires the development of an inter-process communication program to function as an interface between the PLCs and the plant simulation model. This report is the first step to fulfill these needs by developing and validating an emulation methodology for modeling the PLCs within the I&C system. This is in addition to implementing linking the PLC to a transient simulation of a nuclear power plant using a fast-running and robust interface. The validated PLC emulation methodology methods can then be implemented in the SCEPTRE framework and used to emulate the PLCs within the safety I&C system architecture of the nuclear power plant. The developed interface program will be utilized in the NICSim platform to link the dynamic model of the PWR plant to the PLCs within the SCEPTRE framework.

Section 2 - ***Validation of an Emulation Methodology of a Programmable Logic Controller,*** describes the results of the effort on the validation of an emulation methodology of a PLC. A PLC emulation methodology is first developed to characterize the key physical and digital signatures of the PLC and conduct testing to validate these signatures against those of an emulated PLC. The performed validation testing is conducted to determine the settings required to ensure that the performance of the emulated PLC will replicate that of the physical device and replicate its digital network traffic behavior. The developed emulation methodology is first used to create an emulation of a single representative open-source PLC based on a Raspberry Pi 3B+ minicomputer running OpenPLC software. Validation testing is conducted which investigated the physical and digital signatures of the PLC, while linked to a transient process simulation model built using the Matlab Simulink platform (The MathWorks 2018).

Section 3 - *Methods of Interfacing Physics Based Nuclear Power System Model with PLCs,* describes the results on the effort to develop an effective interface to handle the inter-process communication between the validated PLC emulation and a nuclear plant simulation model when implemented together as a combined simulation platform. The developed interface enables the communication of simulation state variables and controller command signals between the PLC and a dynamic simulation model running within Matlab Simulink. This research effort investigates four methods of inter-process communication, including (a) reading/writing to text files, (b) serial communication, (c) TCP communication, and (d) shared memory. The different communication methods are compared to determine which provides the best combination of performance, reliability, and fidelity. The developed interface program is then demonstrated by linking a representative PLC using OpenPLC ladder logic programming with the previously developed DynMo-CBC dynamic simulation model of a space nuclear power system comprised of a gas-cooled fast-neutron spectrum nuclear reactor with Closed-Brayton-Cycle (CBC) energy conversion (El-Genk, Tournier, Gallo 2010). The PLC in the demonstration example is used to start up of the space nuclear power system by controlling the movement of the reactors' control elements.

Section 4 - *Summary and Conclusions* summarizes the results described in this progress report and **Section 5 - *Future Application to NICSim Project*** details how the results of the research described in this progress report will be applied the future tasks in this DOE NEUP project. The next task in this project will employ the developed PLC emulation model to represent the different automatic operation and safety PLC within the nuclear plant safety I&C system, developing the control programming to run on the different PLC within the I&C architecture. A parallel planned task will employ the developed interface to connect the nuclear plant simulation component submodels and instrumentation sensor and control submodels under development in that task with the SCEPTRE framework. The portion of the report describing the validation of an emulation methodology of a programmable logic controller is presented next.

2. Validation of an Emulation Methodology of a Programmable Logic Controller

In this section, an emulation methodology is developed for programmable logic controllers (PLCs) to support future cybersecurity analyses for Industrial Control Systems (ICS) such as the autonomous safety instrumentation and control systems of nuclear power plants. This emulation methodology identifies and characterizes key physical and digital signatures for the selected PLC. The selected digital signature metrics of a PLC include its network response, network traffic, and software versions. The selected physical signature metrics for the PLC include the actuation response time and sampling rate. This work applies this methodology for an open-source PLC implementation consisting of a Raspberry Pi 3B+ minicomputer with OpenPLC to run the ladder logic programming. An extensive validation analysis is performed to characterize the signatures of the real PLC and compare them to those of the emulated PLC.

2.1. Background

Emulated computer systems, also referred to as virtual machines (VMs), are commonly used to perform controlled security experiments in a contained virtual environment in cybersecurity analyses for enterprise IT systems. The Department of Energy's SCEPTRE framework, developed at SNL to enable cybersecurity analyses of ICS, is capable of starting up and handling the virtual network communication using real ICS protocols between large numbers of VMs representing different computers within ICS architectures. While this framework has been used to model digital components of electrical transmission grids and solar power systems, it has not yet been applied to cybersecurity of nuclear power plants. Extending the SCEPTRE framework to modeling the safety instrumentation and control (I&C) systems of nuclear power plants requires the development of emulation models for the PLCs used for autonomous control and the safety system actuation within the plant.

The development of PLC emulation will enable in-depth cybersecurity and physical effect modeling of ICS architectures, such as I&C systems of nuclear reactors without hardware-in-the-loop (HITL) integration. In a HITL setup, a physical PLC is integrated into the digital network being tested. Experiments that use PLCs as HITL are expensive, difficult to scale and change relative to emulated systems. Running an experiment on an entire I&C system architecture, which may include dozens of devices, can become impractical if HITL must be used for every digital device in the plant. A PLC emulation methodology, thus, provides a practical path

forward to study current I&C architectures of nuclear power plants and aid in the design of more secure architectures for future power plants designs.

Most efforts to develop emulation models or virtual machines (VMs) for PLCs have focused mainly on developing virtualization platforms for PLC programming and not virtualization of the computer system. For example, Chunjie and Hui (2009) have investigated developing a PLC VM based on IEC 61131-3 standard PLC programing. This VM was intended to run on the PLCs themselves to create a platform that supports running common programing based on the IEC 61131-3 standard, across different hardware designs. While capable of running PLC control programming, the developed VM was not designed to replicate the characteristics of a specific PLC. Thamrin and Ismail (2011) also developed a VM for PLCs to create a development environment for PLC programming. This VM development environment was planned to help train programmers in the specialized programming languages used by PLCs. The VM also focused on only running the PLC's control programming and not creating a system level virtual machine of a PLC. Gasser (2013) also developed a VM testing environment for using standardized PLC programming across different PLC hardware, similar to Chunjie and Hui (2009). This effort involved extensive testing was performed to characterize the performance of the VM running on the PLC hardware.

None of these prior efforts were aimed at creating an emulated PLC capable of replicating both the physical performance and the digital network behavior of a physical PLC, needed for cybersecurity applications. Thus, the objective of the present work is to develop an emulation methodology for a PLC and establish metrics to validate the developed emulated PLC. This emulation methodology is applied to a representative open-source PLC implementation, and the emulated PLC is validated against the recorded physical and digital signatures of the real, physical PLC hardware. The validated PLC emulation methodology will be incorporated in the DOE SCEPTRE framework to support cybersecurity investigations of I&C system architectures in nuclear power plants. The next section details the developed and validated PLC emulation methodology in this work.

2.2 PLC Emulation Methodology

Table 2.1 outlines the general steps of the PLC emulation methodology. Emulating a PLC implies that the digital and physical behavior of the PLC is reproduced by another system through computational means. The degree of emulation is determined by the project

requirements. A full emulation would reproduce the functionality of the hardware, firmware, operating system kernel, and the software used by the PLC. A partial emulation would replicate only some of these functionalities. Investigations of potential vulnerabilities within software programs or the computer's operating system might only require kernel and software emulation, while investigating potential exploits of vulnerabilities in a chipset's instruction set could require full emulation at the hardware and firmware levels. Several partial emulators of computer systems are available which emulate the kernel and software of a device. Full emulators, however, are far less common due to the drastic increase in complexity.

Table .2.1: Steps within General PLC Emulation Methodology

Step	Action
1	Determine the degree of emulation required
2	Use commercial/Open-source emulation software or develop the emulation as needed
3	Characterize the physical and digital signatures of the PLC to be emulated
4	Benchmark characterized signatures of the emulated PLC against the real PLC using a representative test environment to validate the emulation
5	Make changes within the emulation configuration as needed until the signatures of the real and emulated PLCs agree to within an acceptable margin for the project requirements

The present work requires that the emulated PLC approximate the real PLC, such that the differences between the two systems do not affect the behavior of the connected physics model and could support planned cybersecurity analysis. For the present project, this is accomplished using kernel and software emulation. In addition, the real and emulated systems should be interchangeable and not impacted by the computer hardware running the emulation.

Once a PLC emulation is successfully developed, it is validated against the real system. When obscuring the firmware and the underlining hardware of a PLC, the only way to determine if a PLC is real or emulation is to observe the digital and physical signatures of the device. Table 2.2 shows the signature metrics used to validate the PLC emulation. The selected digital signature metrics of a PLC include the network response, the network traffic, and the software versions. The network response of a device quantifies how the network traffic is transmitted and

received, and determines the rate of data transfer. Similarly, the underlying network traffic for the system determines the type and frequency of the network data packets being transmitted and/or received. Finally, the software versions determine if the exact same software is running on the emulated and real device. If a cybersecurity flaw exists in the software, it should be exploitable on both the real and emulated PLC in exactly the same manner.

Table 2.2: Chosen metrics used to validate PLC emulation

Digital Signatures:
- Network response
- Network traffic
- Software versions
Physical Signatures:
- Actuation response time
- Sampling rate

The selected physical signature metrics for the PLC include the actuation response time and sampling rate (Table 2.2). The actuation response time is that for a PLC to receive data, compute an output, and send an actuation signal. Differing actuation response times would lead to different physical outcomes, by influencing the time history of the physical process. Similarly, the sampling rate of a digital controller would affect the process being controlled by influencing the gain values of complex controller designs affected by the time history of the process. For controllers not using control algorithms that are dependent on the sampling rate, such as a simple switch or trip condition, only the actuation response time would be relevant.

The digital signature metrics are important from a cybersecurity perspective, while the physical signature metrics determines the fidelity of the PLC's response to the connected process. Since PLCs are the interface between the physical and digital world it is paramount that the signatures of the emulated and real PLC be quantified to validate the cyber-physical coupling of the emulation.

2.2.1 Testing Methodology

The developed emulation methodology for PLCs is performed and validated using a representative, open-source PLC architecture consisting of a Raspberry Pi 3 B+ minicomputer running the OpenPLC software, implementing IEC 61131-3 standard programming for PLCs (Alves et al. 2014). The Raspberry Pi is chosen because of the availability of its open source

operating system to create images, the functionality of its on-board digital/analog IO, compatibility with OpenPLC, and the ability to emulate the operating system. The specifications of the real and emulated PLCs are summarized in Table 2.3. VMware emulation software (VMWare 2019) is used to emulate the Raspbian kernel and software for the emulated PLC. Both the real and emulated PLCs run the same operating system and OpenPLC software. To reduce the differences between the Raspberry Pi 3B+ hardware and the PC running the VMware emulation software, one gigabyte of Ram and two processor cores are allocated to the emulated PLC in VMware.

Table 2.3: Real and Emulated PLC Specifications

System	Real PLC	Emulated PLC
Hardware	Raspberry Pi 3 B+	VMware Virtual Machine
CPU	Cortex-A53 64-bit SoC @ 1.4Ghz	AMD FX-8370 64-bit CPU @ 4.3Ghz (2 virtual cores)
Ram	1Gb	1Gb
Operating System	Raspbian Stretch 4.14.79	Raspbian Stretch 4.14.79
Control Software	OpenPLC Version 3	OpenPLC Version 3
Network Interface	Gigabit Ethernet over USB 2.0	Gigabit Ethernet

The used testing environment to validate the PLC emulation against the physical hardware links the real or emulated PLC to a transient simulation model running in Matlab Simulink (The MathWorks 2018) (Fig. 2.1). The Simulink simulation model running on a separate Windows server PC takes the place of the external physical process being controlled by the PLC within the testing environment. An interface program is developed to handle the inter-process communication of data values between the Matlab Simulink simulation model and the remote PLC. The interface communicates state variables generated within the Simulink simulation to a python script using a text file (Fig. 2.1). A second text file is used to transfer the control signals generated by the PLC into the running Simulink simulation. More details on the developed interface with Simulink are discussed subsequently in Section 3.

The python interface program also creates a TCP/IP server to communicate with the PLC, either the Raspbian image running in the VMware emulation or the physical Raspberry Pi. Each PLC uses Raspbian Stretch 4.14.79 to run a client python script which communicates with the python interface program running on the server PC. This client python script communicates the

state variables and control signals to and from the OpenPLC software using the Modbus ICS communication protocol implemented within the PyModbus library (Collins, 2019). The inter-process communication programs used in these analyses are asynchronous with the Simulink simulation, with the python programs running at a rate ~ 10 times that of the transient Simulink model.

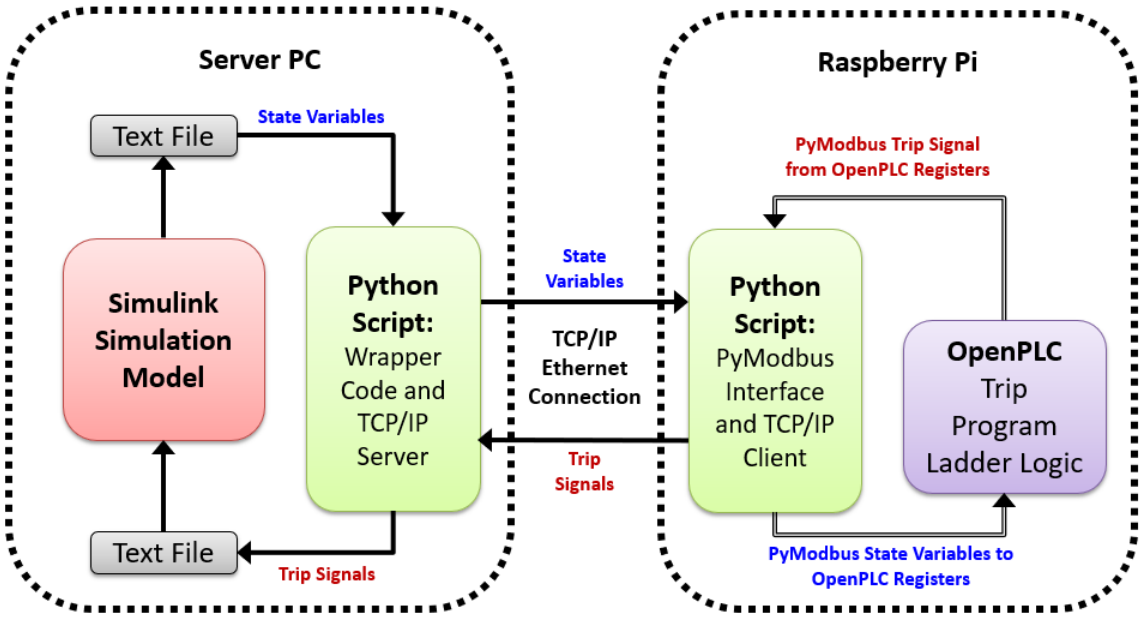

Figure 2.1: Linking block diagram for test system showing data flow between Simulink simulation model and Raspberry Pi PLC using python "wrapper" interface program.

Figure 2.2 shows the components of the test network used for testing and validation. The test network is a simple ethernet network. It consists of a server PC, which is the computer running the Simulink model, a UniFi switch, and a PLC. For each experiment, only one PLC is connected to the UniFi switch at a time. The Simulink simulation model within the benchmark testing environment generates a sine wave, Eq. 2.1, which provides state variable input to the PLC as:

$$f(t) = \sin\left(\frac{\pi}{2}t\right) + 1. \tag{2.1}$$

The sine wave runs in sync with real time, is outputted by the server PC and sent to a switch to be routed to the PLC. An isolated testing network with a managed switch was used to eliminate network routing differences during the testing between the real and emulated PLC. The local Dynamic Host Configuration Protocol (DHCP) server handling the IP addresses for the isolated

network was run on server PC. The Wireshark (Wireshark 2019) utility was used to capture the network data traffic between the server PC and real or emulated PLC.

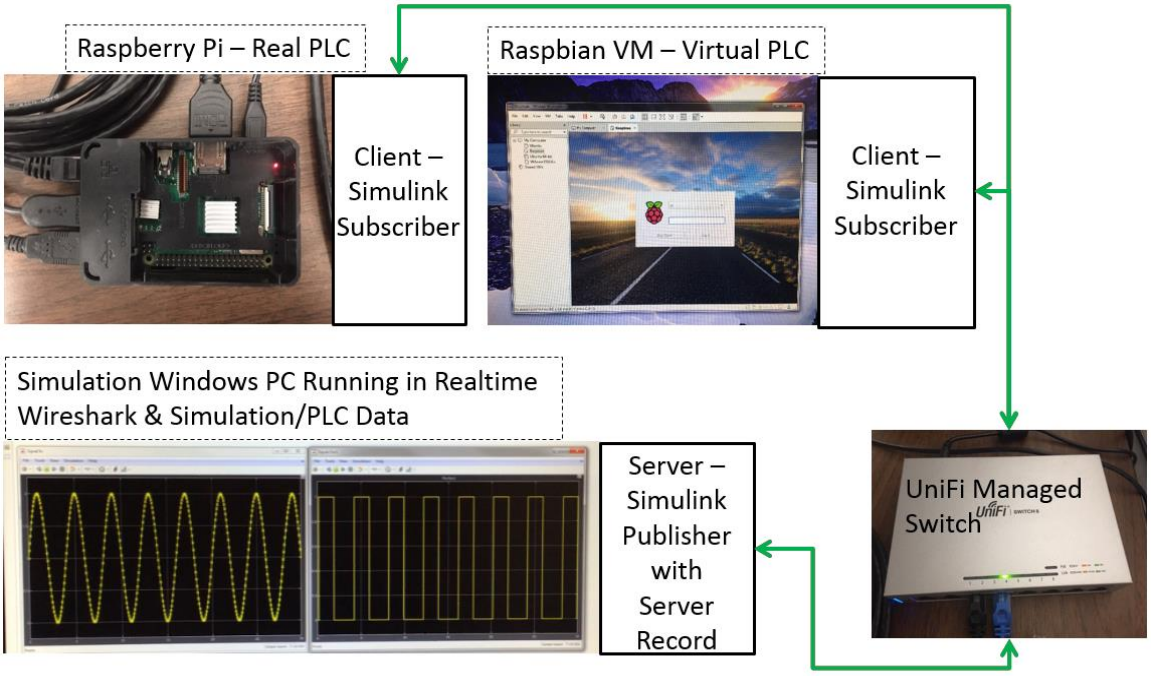

Figure 2.2: Overview of real and emulated PLC comparison experimental setup.

When the PLC receives a new value from the server, the implemented ladder logic programming in OpenPLC determines if the sine wave is above or below the set point. Using a set point value of 1.2, results in a trip signal value of one if the sine wave value greater than the set point, and a value of 0 if it is below the set point. With the generated sine wave this should result in a trip signal period for 1.74 seconds followed by a null period of 2.26 seconds for each wave. The total simulation length for each run was 300 s. Except when noted the OpenPLC program running on either the emulated or real PLC used a sampling period of 50 ms. The actuation response of the PLC is routed back the server PC in the opposite direction creating the square wave seen in the lower left of Fig. 2.2.

2.3. Validation Results of the PLC Emulation Methodology

The following sections describe the results of the validation testing characterizing the signatures of the physical PLC and comparing them against the developed PLC emulation. First in Section 2.3.1, an analysis is performed investigating the real-time condition of the linked Simulink simulation and PLC. Sections 2.3.2 and 2.3.3 present the validation testing of the digital signatures of the emulated PLC. Section 2.3.2 presents the results of the network response

comparison and Section 2.3.3 shows the results comparing the network data packet traffic. Sections 2.3.4 and 2.3.5 present the validation testing for the Physical Signatures, showing the results of the actuation response time and sampling rate, respectively.

2.3.1 Real-Time Condition

The simulation model runs in sync with real time using Simulink Real-Time Desktop (The MathWorks 2018) to ensure that both the real and emulated controllers receive the same values from the simulation at the same rate. Real-Time synchronization is required for comparing the real PLC and Emulated PLC. If the simulation process operates at a different rate, it will affect the actuation response time of the controller. For example, if the simulation for the real PLC runs at a slower rate, it is possible that the real PLC will record a faster actuation response time relative to the emulated PLC. Real-time synchronization helps ensure that the rate and exact time of the simulation response from the different PLCs are not affected by time differences in the simulated process.

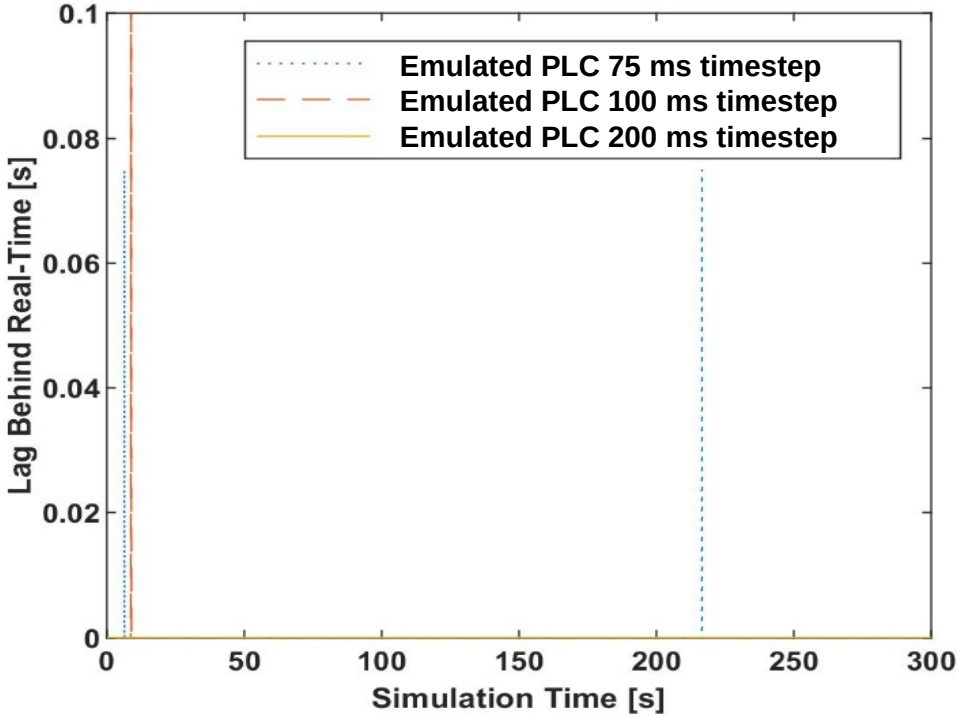

Figure 2.3: Emulated PLC real-time sync lag.

To investigate how the timestep size impacted the real-time synchronization, Simulink simulations are run with major time steps of 75, 100, and 200 ms. The real time tolerance within the Simulink Real-Time Synch feature is set so that if more than one CPU tick is missed in a row, the simulation exits and the experiment is restarted. Figs. 2.3 and 2.4 show that both the

emulated and real PLCs produced a small number of deviation events from being in sync with real time. The simulations that used a major time step of 200 ms, however, did not experience any deviation from real time.

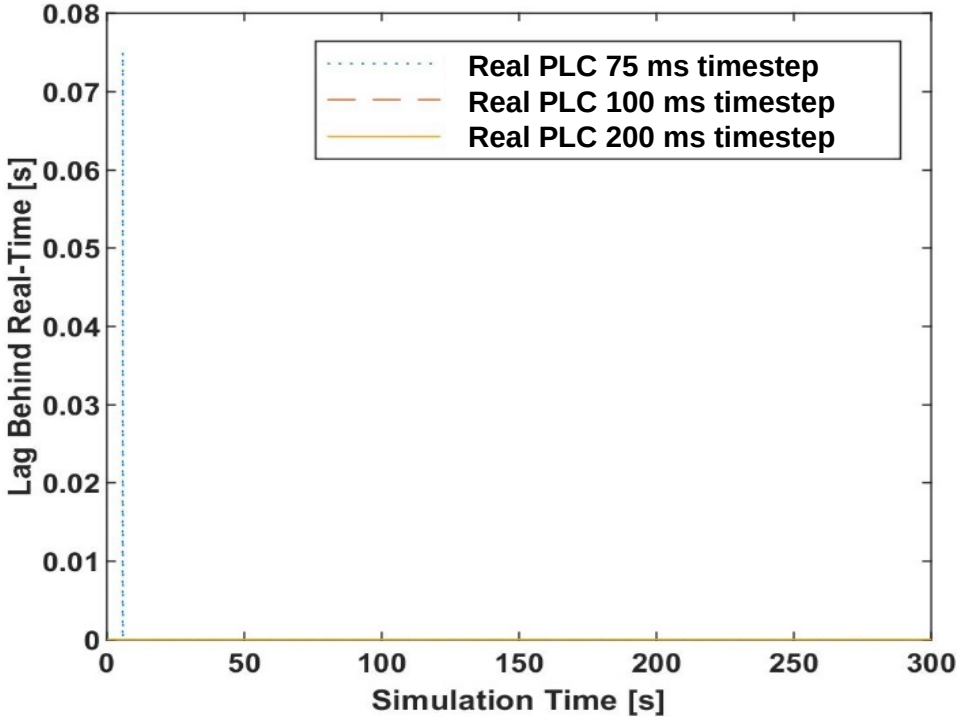

Figure 2.4: Real PLC real-time sync lag.

A deviation from real-time causes the controller to miss a timestep, and the error catching routine makes the controller use the previous values until the next timestep updates the state variables. A missed data point thus delays the response of a controller by one timestep. The data for the emulated PLC using 100 ms major time step recorded a single deviation at time = 8.7 s (Fig. 2.3). Two deviations occurred for the emulated PLC using 75 ms major time steps at simulation times = 6.3s and 216.4s. The results for the real PLC recorded a single deviation at time = 5.7s when using a major time step of 100 ms (Fig. 2.4). For the simulations with timesteps of 75 and 100 ms, deviations from real-time occurred less than 0.1% of all simulation time steps.

For the most realistic simulation of a PLC controlling a physical process, the major fixed timestep of the simulation needs to be much less than the response time of the PLC running in sync with real-time. When the simulation is running slower than the response time of the PLC, the fidelity of the simulated process is low. For validation of the testing setup, using timestep

sizes of 75 and 100 ms result in a few deviations from synchronization, while for a timestep of at least 200 ms, no deviations are observed.

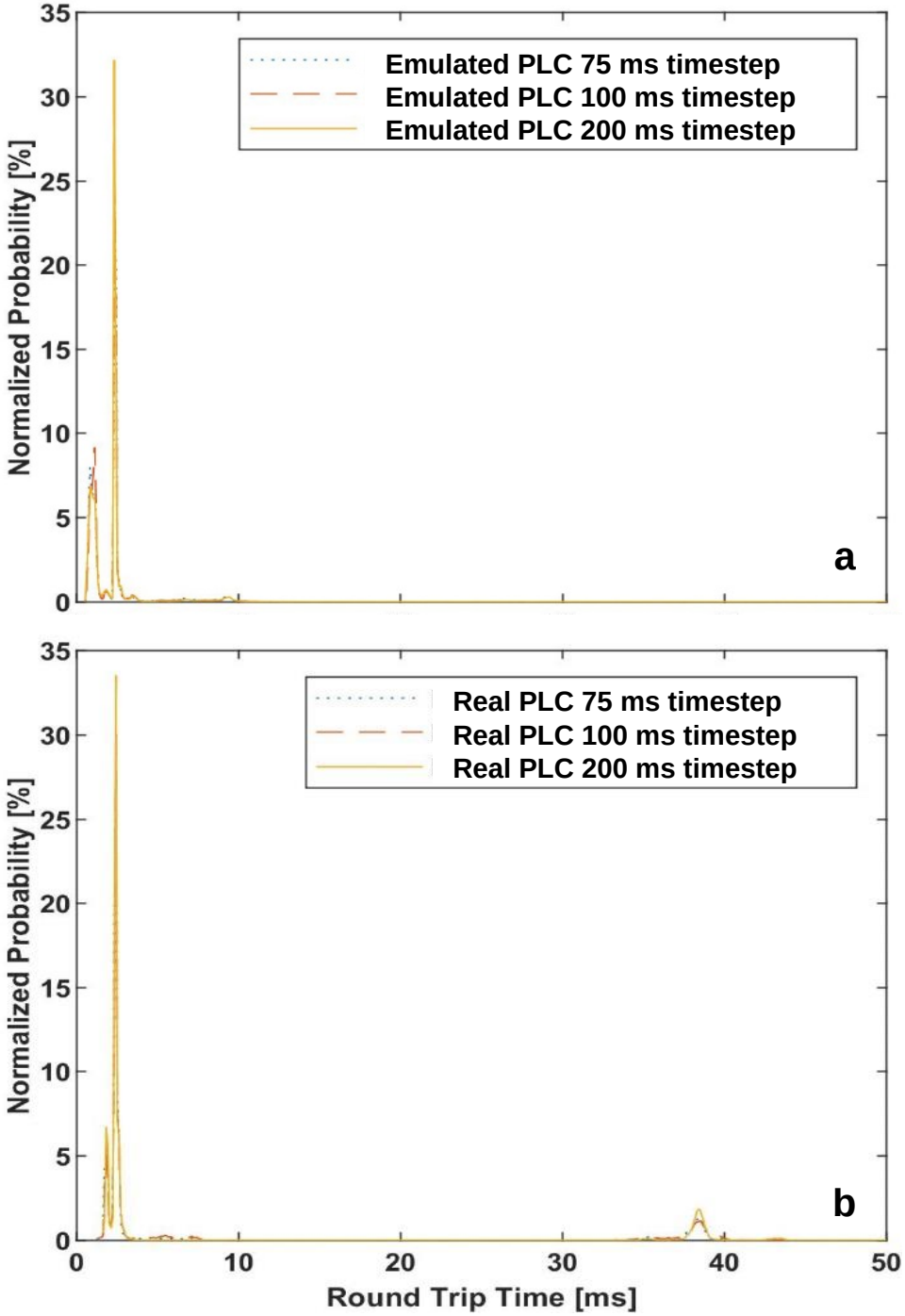

Figure 2.5: Round trip time of TCP packets for the a) emulated PLC and b) real PLC.

2.3.2 Network Response

The network response characteristics of the emulated and real PLCs are collected using Wireshark for packet capture (PCAP) of the network communication between the server PC and

the PLC. The PCAP software is run on the server PC for all simulation cases. The network response of the emulated and real PLCs is evaluated by comparing the TCP packet round-trip time and the network statistics for the TCP/IP socket communication between the Server PC and real or emulated PLC. The network round trip time is important because the slowest exchange of TCP packets determines the physical limit of outputting the data by the simulation to the PLC. Therefore, the lower limit of the sampling time for the PLC is that of the slowest communication of data from the server to the client. In other environments, the communication method may be different from the TCP/IP communication used here, but the underlining principle of quantifying the rate at which data can be transferred to quantify the network response remains pertinent.

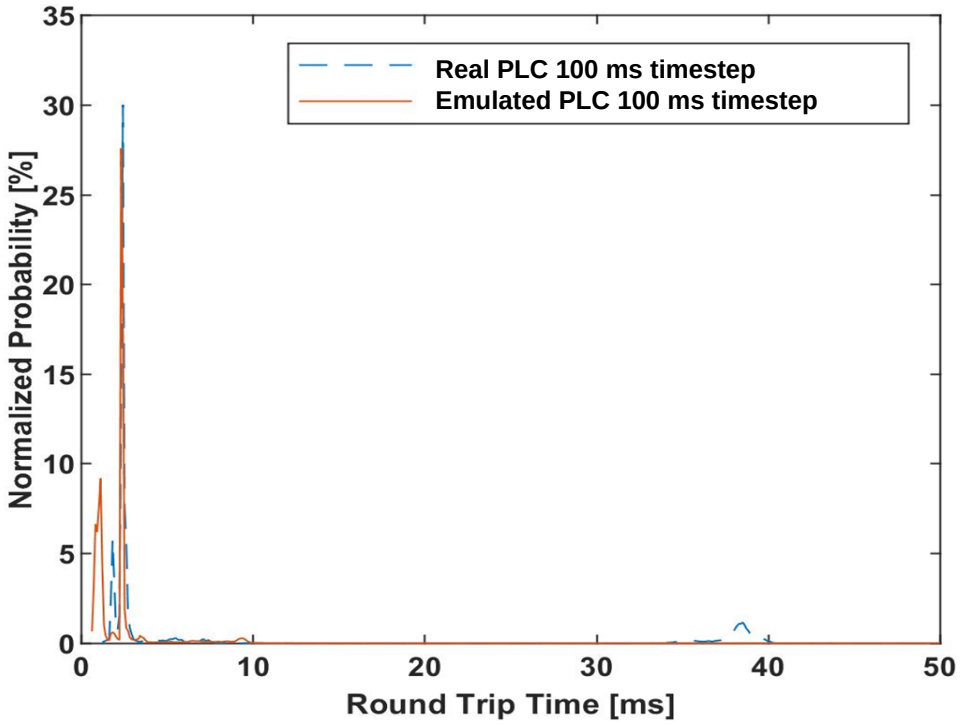

Figure 2.6: Comparison of the round-trip time of TCP packets for the Real and Emulated PLC

The network response of the emulated PLC is shown in Fig. 2.5a, and the network response of the real PLC is shown in Fig. 2.5b. Simulation cases are run with major timesteps of 75, 100, and 200 ms to evaluate the effect of the simulation timestep on the network response. It is observed that the simulation time step does not affect the network response for both the real and emulated PLCs (Fig. 2.5a-b). Both controllers exhibit two spaced peaks in the probability distribution of the TCP packet round trip time between 0 and 5 ms. Nearly all packets fall within this time range when using the emulated PLC. In contrast, the real PLC has a small number of

slower TCP packets, which make up a distinct second cluster of packets that are transferred at a rate between 35 ms and 45 ms.

Figure 2.6 directly compares the real and emulated network traffic for a Simulink simulation time step of 0.1s. The most probably round trip time of the emulated and real PLCs is the same, with peaks clearly overlapping (Fig. 2.6). The second of the two peaks in the 0-5 ms range in the probability distributions for the two PLCs occurs at a slightly shorter round-trip time for the emulated PLC compared to the real PLC (Fig. 2.6). The peak representing the group of slower packets is seen in the data for the real PLC, but not by the emulated PLC.

Figure 2.7: Raspberry Pi units used in the network response identification tests. Raspberry Pi number 1 is the same unit which was used in the real and emulated PLC comparison.

The testing using multiple Raspberry Pi's is used to determine if the small observed group of packets of times from 35 ms and 45 ms is unique to the specific Raspberry Pi initially used for benchmarking, or it is characteristic to the that computer model. Three, identical Raspberry Pi 3 B+ (Fig. 2.7) units are used to investigate the network response of the Raspberry Pi. The same setup in the PLC comparison is used to measure the network response of the three separate Raspberry Pi units. A simulation fixed time of 100 ms is used in this experiment. To avoid any discrepancies between the pace of the simulation, all experiments are run using Simulink Real-Time and the tolerance is set to one missed CPU tick. Deviations from real-time sync are negligible; it is also observed that the deviations from real-time sync corresponded with a missed data point over a 300 s simulation run time.

Since all three Raspberry Pi units had the same simulation conditions, input, and actuation response, the normalized packet round-trip time in Fig. 2.8 are compared directly. The captured network data for all three units has the same TCP packet round trip time distribution, including the isolated peak of slower data packets of times of 35 ms to 45 ms. These results suggest that

this cluster of relatively slow packets is a characteristic of the Raspberry Pi 3B+. The same effect is not observed on the emulated PLC using the network card of the computer facilitating the emulation.

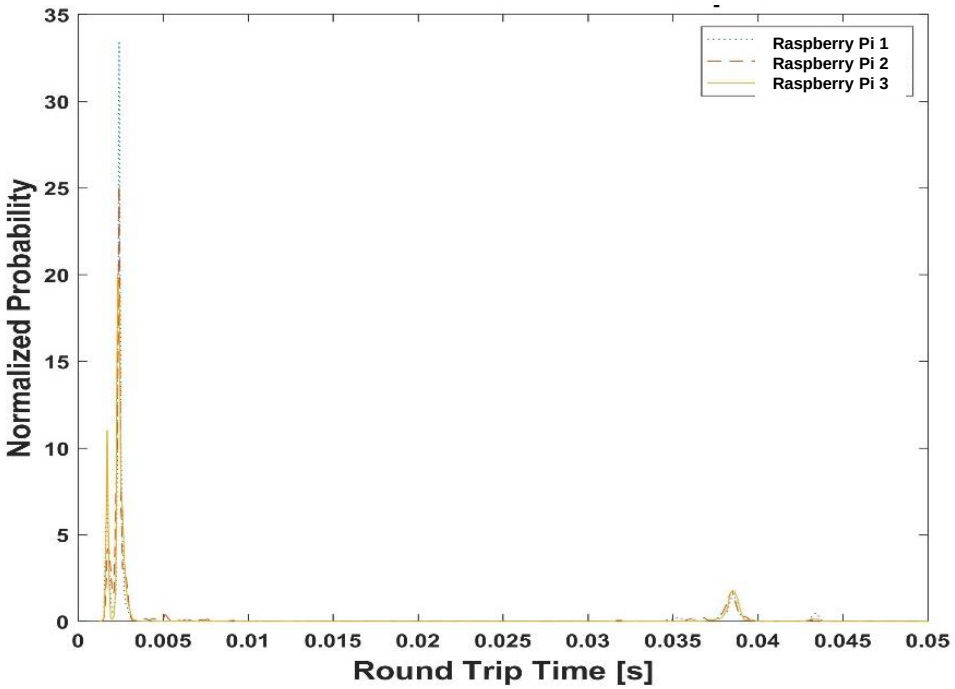

Figure 2.8: Normalized packet round-trip time distribution for Raspberry Pi units 1-3.

In addition to the packet round trip time, the data throughput of the emulated and real PLC is captured, and the statistics are summarized in Table 2.4. The emulated PLC has a significantly higher data transfer rate relative to the real PLC. This difference is due to the fact that the hardware used by the emulated PLC has a higher bandwidth than the real PLC. The results in Figs. 2.5 and 2.6 show, however, that higher bandwidth does not result in faster communication speeds in the validation testing setup. For both the emulated and real PLCs the majority of the packets sent during the experiment had communication speeds faster than 10 ms.

2.3.3 Network traffic

The collected packet capture data is analyzed to determine the types of packets sent during the experiment. The data for all experiments was combined for the real and emulated PLC to decrease the differences in the time of day each experiment is ran. It is possible that some programs will send out network communication based on the time of day or the current state of the operating system. The numbers and categories of data packets collected by the Wireshark utility are given in Table 2.5 for the emulated PLC and Table 6 for the real PLC tested. The vast

majority (> 98%) of the collected data packets are TCP packets (Table 2.5). The fraction of the total packets collected in each category between real and emulated PLCs differed by < 1%.

The network traffic signatures for the emulated PLC are determined to be comparable to the real PLC with the exception of the bandwidth. The collected network packet data in Table 2.5 shows, however, that the emulated PLC is able to use the same network protocols and generate packets of the same data types and with similar proportions to the total level of network traffic as the real PLC. These similarities in types and proportions of network traffic are important for the capability of the emulated PLC to represent the real PLC in cyber-security investigations.

Table 2.4: Captured data transfer rate for the emulated and real PLC

Item	Data for Emulated PLC / Real PLC		
Measurement	0.075s Time Step	0.1s Time Step	0.2s Time Step
Packets	131,354 / 35,952	132,544 / 34,126	133,548 / 35,419
Time Span, s	295.63 / 296.92	295.63 / 295.52	295.63 / 294.71
Average Packets Per Second	444.3 / 121.1	448.3 / 115.5	451.5 / 120.2
Average Packet Size, Bytes	71 / 72	71 / 72	71 / 72
Average Bytes/s	31 k / 8.7 k	31 k / 8.3 k	31 k / 8.6 k

2.3.4 Actuation Response Time

An important physical signature to measure for quantifying the performance of a PLC is the actuation response time. The actuation response time is defined here as the time it takes for the PLC to execute a control action based on the inputs being sampled from the process being controlled. To determine if the observed actuation response deviates from an ideal response, internal logic in the Simulink simulation model is used to generate a ‘true’ response signal. This ideal ‘true’ signal is recorded internally by Matlab Simulink and not dependent on the signals transmitted to and received from the PLC.

Thus, the ideal response represents a control action with zero time lag relative to the process simulation. The ideal response is compared against the generated responses of the real or emulated PLC to quantify any time lag in response time due to network communication or differences in PLC computational time. First, the state variable signal received by the PLCs is compared to the original signal generated by the Simulink model to verify that the real and

emulated PLCs are receiving the same and correct input data. Fig. 2.9a-c shows the result for simulation major timesteps of 75, 100, and 200 ms. For each case, the OpenPLC program on the real and emulated PLC operates at a sampling rate of 50 ms. The results show that the real and emulated PLCs receive the same Sine wave signal (Eq. 2.1) generated by the Simulink model with the lines falling on top of one another. This excellent agreement is consistent for the three Simulink simulation timestep values tested (Fig. 2.9a-c).

Table 2.5: Wireshark network PCAP data results for different packet types and their relative proportions for the emulated and real PLCs

Packet Data Type	Emulated PLC percent of total packets (number of packets recorded)	Real PLC percent of total packets (number of packets recorded)
ARP	0.14%, (554)	0.66%, (690)
BROWSER	0.00%, (6)	0.02%, (20)
CLASSIC-STUN	0.02%, (60)	0.05%, (59)
DHCP	0.01%, (23)	0.00%, (0)
DNS	0.10%, (388)	0.00%, (0)
HTTP	0.05%, (213)	0.21%, (228)
ICMP	0.00%, (4)	0.00%
ICMPv6	0.00%, (9)	0.00%
LANMAN	0.00%, (8)	0.00%
LLDP	0.01%, (32)	0.03%, (30)
LLMNR	0.01%, (37)	0.01%, (8)
MDNS	0.00%, (2)	0.00%, (0)
NBSS	0.00%, (4)	0.00%, (0)
SMB	0.01%, (24)	0.00%, (0)
SSDP	0.02%, (77)	0.03%, (28)
STP	0.11%, (444)	0.42%, (449)
TCP	99.46%, (395,316)	98.56%, (103,984)
UDP	0.00%, (16)	0.00%, (0)
NBNS	0.06%, (222)	0.01%, (11)
NTP	0.00%, (7)	0.00%, (1)
DHCPv6	0.00%, (7)	0.00%, (0)

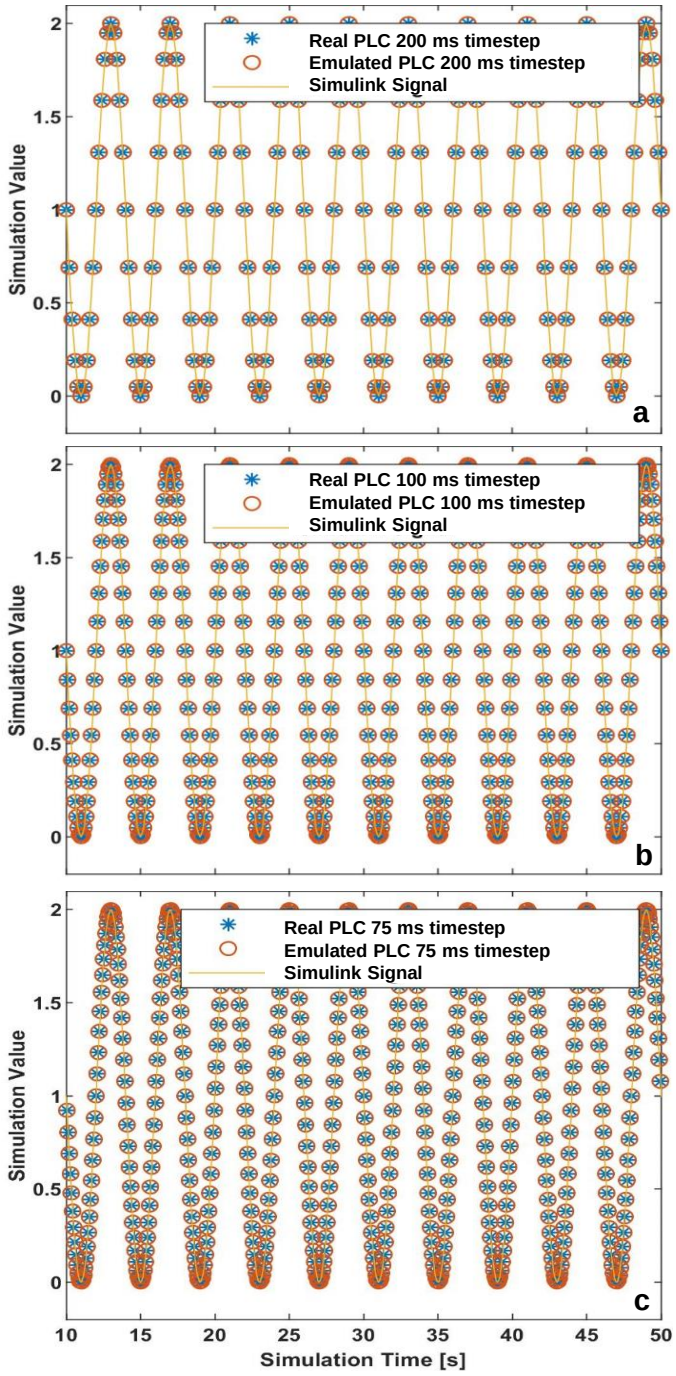

Figure 2.9: Data input to the real and emulated PLC compared to the sine wave generated by Simulink for major time steps of (a) 200 ms, (b) 100 ms, and (c) 75ms.

Once the PLC receives the sine wave input, the ladder logic implemented in OpenPLC determines if the current Sine wave values is above or below the setpoint of 1.2. If the Sine wave value is above 1.2, the command output is a value of one, and if it is below 1.2, the output is zero. Therefore the command signal creates a square wave signal over time. The controller

response signal transmitted by the connected PLC is recorded within the running Simulink simulation and compared to the ideal baseline signal to identify the response time lag. This time lag in the response of the PLC is unavoidable in this asynchronous process. For example, if the simulation jumps from ten percent below the set point to ten percent above the set point, from a step to the next, the PLC would not actuate until the set point is exceeded. The actuation response for the emulated and real PLCs are shown in Fig. 2.10a-c and Table 2.6 for the different simulation major time steps. For ease of viewing, only 40 s of the controller response is plotted in Fig. 2.10 out of the 300 s total simulation run time. The controller response for both the emulated and real PLCs consistently lags behind the ideal response signal, with the square wave signal for the PLCs visibly shifted to the right (Figs. 2.10a-c).

The actuation response statistics are summarized in Table 2.6 for the major simulation timesteps of 200, 100, and 75 ms. Table 2.6 shows the recorded time lags observed for each square wave signal and the percentage of actuation signals which featured that degree of time lag relative to the ideal response signal. The recorded lag in the actuation response for the different timesteps is discrete, and ranges in size from 2-4 major timesteps (Table 2.6). The actuation response using 200 ms major time step size is nearly the same, with the emulated PLC actuation response lagging very slightly behind that of the real PLC.

Decreasing the major time step size to 100 ms increases the divergence between the emulated and real systems, with the emulated PLC responding slightly faster on average compared to the real PLC (Table 2.6). This trend continues as the timestep further reduced to 75 ms, where the difference between the real and the emulated PLC is further increased. The network response analysis for the real PLC suggested that the slowest roundtrip communication is around 50 ms, not accounting for computational time. It is expected that as the real PLC approaches the physical limit it can receive information, the actuation response time will diverge from the ideal response rate. A similar physical communication limit also exists for the emulated PLC, but it appears to be lower than that of the Real PLC due to its higher data transfer rate.

As noted in Section 2.3.1, the inter-process communication programs in these analyses are asynchronous with the Simulink simulation. This is for better validation comparison between the real and emulated PLCs. The delay in the actuation response signal generated by the OpenPLC programming can be characteristic to the PLCs computing hardware and software. A synchronous inter-process communication method could be employed to eliminate the time lag

between the observed and ideal actuation response signals by ensuring that the Simulink simulation will wait for the PLC control signal before continuing to the next timestep. The minimum size of the simulation timestep to be used, however, will still be limited by the network response time for the communication between the PLC and server PC running the process simulation model.

Table 2.6: Actuation response for the real and emulated PLC with differing simulation timesteps

Major Time Step == 200 ms, Simulation Time == 300 s		
Lag	Real PLC Actuation Response Time	Emulated PLC Actuation Response Time
400 ms	98.67%	96.67%
600 ms	1.33%	2.67%
800 ms	0.00%	0.67%
Sample Size	35,177 TCP Packets	133,855 TCP Packets
Major Time Step == 100 ms, Simulation Time == 300 s		
Lag	Real PLC Actuation Response Time	Emulated PLC Actuation Response Time
200 ms	93.33%	96%
300 ms	5.33%	3.33%
400 ms	0.67%	0.67%
600 ms	0.67%	0.00%
Sample Size	35,177 TCP Packets	133,855 TCP Packets
Major Time Step == 75 ms, Simulation Time == 300 s		
Lag	Real PLC Actuation Response Time	Emulated PLC Actuation Response Time
150 ms	80.67%	95.33%
225 ms	19.33%	4.00%
300 ms	0.00%	0.67%
Sample Size	35,177 TCP Packets	130,288 TCP Packets

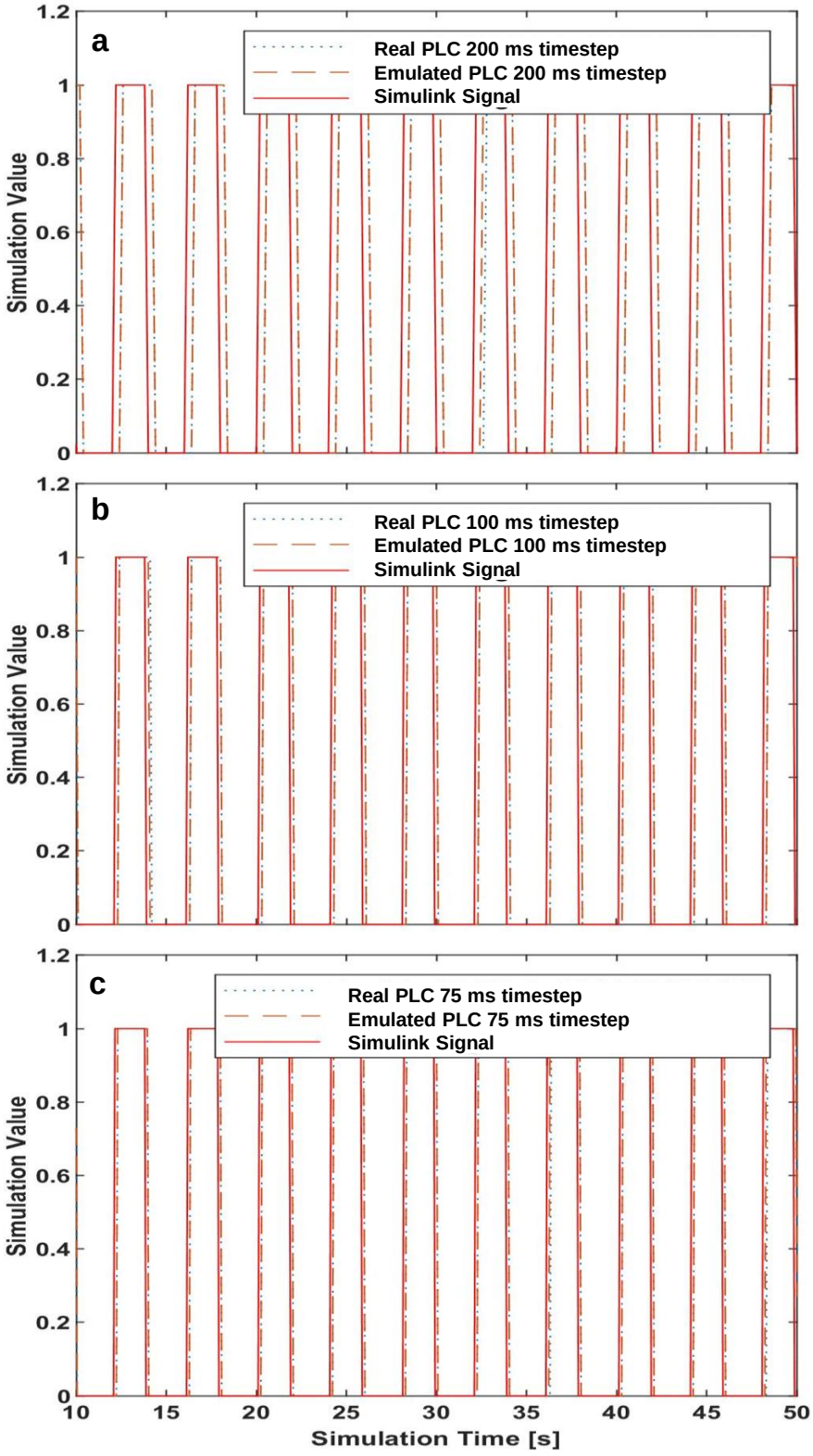

Figure 2.10: Actuation response of the real and emulated PLC compared to the ideal response generated by Simulink for major time steps of a) 200 ms, b) 100 ms, and c) 75 ms.

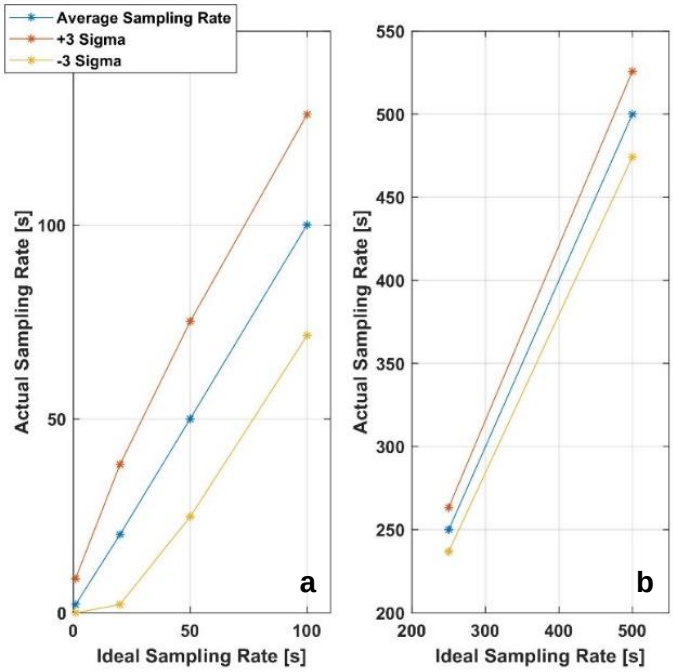

Figure 2.11: Recorded actual sampling rates of emulated PLC compared to ideal sampling rate

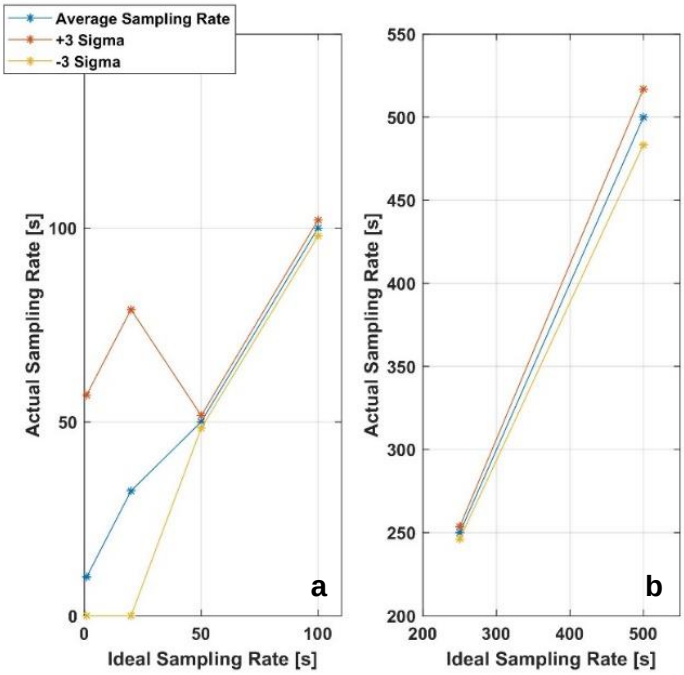

Figure 2.12: Recorded actual sampling rates of the real PLC compared to the ideal sampling rate

2.3.5 Sampling Rate

The rate at which a PLC samples the controlled process is a very important parameter based on classical control theory, and in terms of determining if the PLC is fast enough to monitor the

physical process in questions. The sampling rate of the real and emulated PLC is measured using an algorithm implemented in python that writes a value to a register, get the wall clock time at the end of the ladder logic script execution, write a new value to the same register, and time how long it takes the PLC to submit a new control action based on the new input. This method assumes that the python script is running at a much faster rate than the ladder logic program and that the computational time required to get the current time is negligible relative to the sampling rate. These are valid assumptions for the current testing setup. The real and emulated PLCs are tested with sampling rates specified for the OpenPLC program of 1, 25, 50, 100, 250, and 500 ms, and the actual sampling rate recorded using the python algorithm to check for any deviations from the ideal specified rate.

Figures 2.11 and 2.12 present the results of the sampling rate analyses for the emulated and real PLCs, respectively. The sampling rate results for 1 – 100 ms are shown in Figs. 2.11a and 2.12a, with the results for sampling rates of 250 and 500 ms are shown in Figs. 2.11b and 2.12b. The actual and ideal sampling rates are seen to be equivalent for both the emulated and real PLCs when the OpenPLC sampling rate was set to a value ≥ 50 ms. Below 50 ms the emulated PLC only slightly diverged from the ideal rate, while the real PLC is no longer capable of matching the ideal sampling rate. Comparing the variance in the actual measured sampling rate, it can be seen that for sampling times above 50 ms the real PLC experienced less variance than the emulated PLC. The difference in the variance between the emulated and real PLCs decreased as the sampling rate increased, with the smallest variance observed at a sampling rate of 250ms.

These results for the sampling rate analyses suggest that the emulated PLC matches the same average sampling rate as the real PLC. This is true as long as the sampling rate specified for the OpenPLC programing is set to ≥ 50 ms. The recorded sampling time includes the network communication between the server PC and the PLC, thus the faster sampling rates run into the network communication time observed between the PLC and server PC running the Simulink simulation model. The emulated PLC is capable of supporting faster communication rates than the real PLC hardware (Table 2.4). For sampling rates < 50 ms the emulated PLC did not divergence as seen in the data for the real PLC (Figs. 2.11-2.12).

2.4. Summary

This work developed a PLC emulation methodology and applies it to validating an emulation of a Raspberry 3B+ as the benchmarking hardware for a PLC running the open-source OpenPLC

program. The chosen PLC emulation uses the VMware software to emulate the kernel and software of the physical device and employs the same OpenPLC control program software and network communication protocols as the real PLC. The validation testing analyses investigates the emulated and real PLCs linked to a transient process simulation model running within Matlab Simulink. Python scripts running on the server PLC with the Simulink simulation and PLC are used to handle the inter-process communication between Simulink and the OpenPLC control program running on both the emulated and real PLCs.

The validation testing analyses have shown that the real and emulated PLCs, if properly configured, can have digital and physical signatures that approximate each other. Using the same software on both systems should mean that any vulnerabilities in the OS kernel, programs, and communication protocols used on the real system would be reflected in the emulated system and that the internal logic programming is highly consistent between the two.

The testing effort characterized the digital signatures of the network response and types and proportions of network traffic recorded between the PLC and server PC running the Simulink process simulation. The analyses of the network response showed that the emulated PLC has a much higher bandwidth capacity relative to the real PLC, however this was found to not have a significant impact on the relative communication speeds relative to the real PLC. Setting the major simulation time step greater than the slowest packet round-trip time of ~50 ms resulted in there being a negligible difference in the network response time between real or emulated PLCs.

The difference in bandwidth did have an effect on the recorded network traffic, with the emulated PLC transmitting approximately three times as many total data packets as the real PLC. Although the total number of packets was found to be different, the PCAP analyses showed that the emulated PLC generated the same variety of network traffic packets as observed using the real PLC. When looking at the relative proportions of the data packets in the different categories collected by the Wireshark utility, the real and emulated PLC did not have a difference of greater than 1% in any data packet type.

The physical signatures of the actuation response time and sampling rate are also characterized for the real PLC and compared against the developed PLC emulation. The validation testing analyses showed that the actuation response times for the emulated and real PLCs converge as the simulation major time step increases. For a simulation major time step of 200 ms the difference between actuation signal response times of the real and emulated PLC was

found to be within 2%. This suggests that future I&C experiments with time history dependent processes should use a Simulink simulation time step ≥ 200 ms to make sure that the PLC emulation response exactly matches the physical Raspberry Pi 3 B+.

Finally, validation analyses investigating the sampling rate the OpenPLC programming found that a sampling rate ≥ 50 ms resulted in both the real and emulated PLCs matching the specified sampling time set in OpenPLC. For sampling rate < 50 ms the real PLC observed significant deviation in the actual recorded sampling rate from the ideal value. The smallest difference in the sampling rates of the emulated and real PLC systems occurred for a sampling rate of 250 ms.

The validation testing analyses demonstrate PLC emulation with actuation responses and sampling rates which are essentially indistinguishable from the real PLC to the process simulation being controlled. From a cybersecurity perspective, this PLC emulation runs the same software, communicates using the same communication protocols, and generates the same types and proportions of network traffic data types as the real device. Employment of the developed PLC emulation methodology shows the importance of selecting the proper configuration parameters to ensure that the emulated PLC behaves comparable to the real system, confirming the need for detailed characterization and comparison of the physical and digital signatures as is performed in this work.

3. Methods of Interfacing Physics Based Nuclear Power System Model with PLCs

A timely and accurate data and control transfer interface is developed to allow for inter-process communication between a simulation model running in Matlab Simulink and an external programmable logic controller (PLC). This interface is demonstrated by connecting a dynamic simulation model of a space reactor power system with multiple closed Brayton cycle energy conversion loops (DynMo-CBC) and a Raspberry Pi PLC. This work investigates four data transfer protocols (text file transfer, serial communication, shared memory, and TCP/IP communication) to determine the most effective option for communication between Matlab Simulink and external PLCs.

3.1 Background

Digital instrumentation and control (I&C) systems can offer higher efficiency and performance in process control applications compared to analog systems, but at the cost of possible cyber security risks. The increasing integration of digital I&C systems in nuclear power plants creates potential cyber security risks which could present a serious threat to national security. The nuclear instrumentation and control simulation (NICSim) project aims to develop a platform to investigate the cyber security of nuclear I&C architecture which links emulated and simulated components representing the I&C system of a nuclear power plant with a physics-based simulation model of the plant for direct control feedback. Developing this platform requires an interface to handle the inter-process communication between the plant simulation model and the emulated PLC in the I&C system.

Terrestrial power reactors systems are not the only systems at risk, space nuclear power systems are also vulnerable. Future space nuclear power systems are expected to utilize digital I&C systems which will communicate back to earth via radio transmission. Cyber-attacks are not the only vulnerability to space reactor power systems; micro meteoroids, radiation, and component failures can impact operation and require a timely response by the onboard controller. Having an onboard simulation of the reactor power system to compare operating parameters with those of the real system, in real time, would allow identifying any change in operational condition and determine a correction and adjustment of system operation. This requires an effective interface of the simulations with the digital I&C systems, in a timely and accurate manner, to compare the simulated operational parameters to those of the real system.

The commercial nuclear plant simulation model in the NICSim platform is constructed within the modular and versatile Matlab Simulink platform (MathWorks 2018). Matlab Simulink is a powerful simulation tool commonly used in industry to simulate complex physics-based systems. It is selected for its flexibility for meeting adverse physics-based simulation needs. Simulink does not have a native solution for timely and accurate data transmission interfaces with external control systems, such as PLCs, that can be compiled inline by the Simulink C coder into an executable. An executable is desirable for running simulations outside of the Matlab platform. This is important when needing to run simulations on server high performance computer networks without a Matlab license.

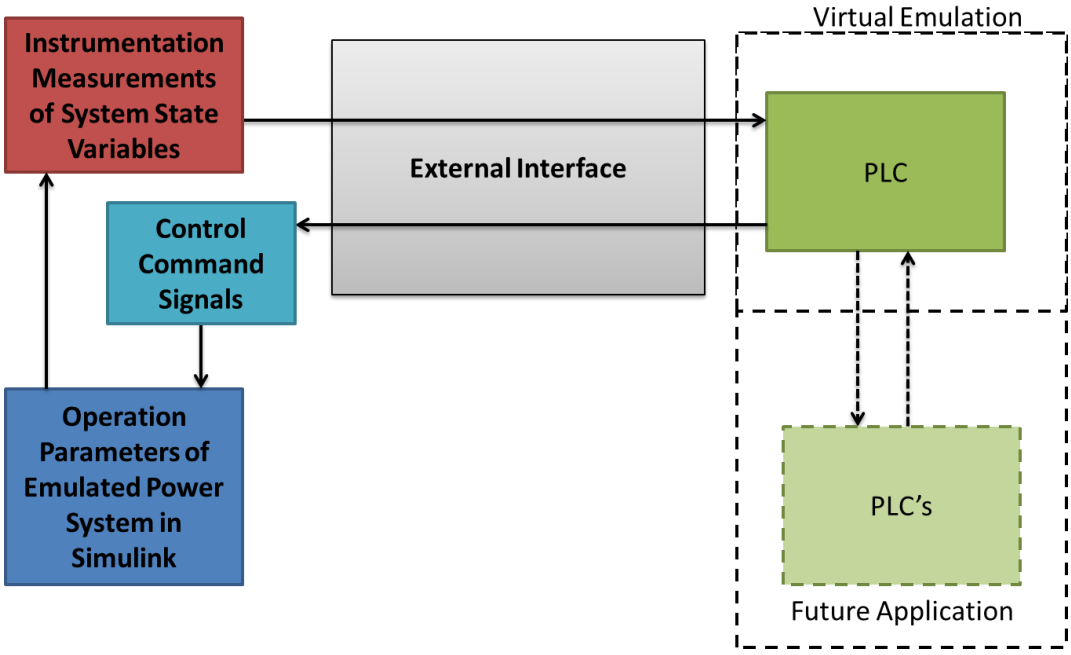

Figure 3.1: Flow diagram of the functionality of the developed external interface.

Many third-party solutions have been created to connect Simulink to external controllers; however, these have only been developed for specific proprietary controller software. Siemens has developed application programming interfaces (APIs) to connect their SIMIT product to Simulink (Siemens 2018). However, this product cannot provide a light weight adaptable interface. Similarly DIgSILENT has a Simulink interface that is designed only for their PowerFactory product, but again it lacks flexibility (Kerahroudi et. al. 2014). Therefore, there is a need to develop a general, lightweight external interface to transport data out and transfer control signals into Simulink from physical or emulated PLCs. This interface can compile in-line

with the Simulink simulation model into a stand-alone executable.

Figure 3.1 depicts the functionality of the external interface developed herein to facilitate timely and accurate transfer of data and control signals. Any delay or inaccuracy in data and control transmission can negatively impact the simulation and controller response. Therefore, the interface must support fast and reliable inter-process communication, bridge the communication gaps between Simulink, PLCs, and computer networks, and capable of connecting with multiple PLCs across I&C system network architecture. Flexibility of the interface is required to accommodate potential simulated systems and communication protocols.

3.1.1 Objectives

The objective of the effort described in this section is to develop a fast and reliable data and control interface, and investigate multiple communication methods, These methods are benchmarked to ascertain which best facilitates the most timely and accurate communication. Once selected, the functionality of the best interface method will be demonstrated by connecting an existing dynamic simulation model of a space reactor power system to an external programmable logic controller (PLC). This is the open-source PLC used in section 2 with OpenPLC running on a Raspberry Pi 3B+ minicomputer (Alves 2019). The dynamic simulation model of the space reactor power system, with multiple closed Brayton cycle loops (DynMo-CBC), has been developed by the University of New Mexico's Institute for Space and Nuclear Power Studies (UNM-ISNPS) using the Matlab Simulink platform (El-Genk, Tournier, and Gallo 2010). This section describes the control of this power system using the Raspberry Pi PLC. The results will demonstrate the capabilities of the developed data and control interface to support linking a PLC with a dynamic model of a commercial nuclear power system model to be developed in the NICSim project.

3.2. Approach

Matlab Simulink is a powerful tool for physics-based modeling. However, it offers only limited native methods of external communication and control. It includes a powerful tool to implement C code within a model, referred to as S-Functions. These functional blocks allow C code written in a Matlab specific structure, to run and compile in-line with Simulink models (The MathWorks 2019). These S-Functions are structured to allow the C code to interact with the Simulink model during specific operational steps performed by the Simulink solver in the simulations, such as the initialization, output, update, and the termination phases.

Initialization occurs before the simulation model begins, while the termination phase happens at the end of the simulation. The code in the output phase operates during every minor simulation time step, where Simulink is solving for the parameters of the physics-based models, iteratively till solution convergence. The code in the update phase operates near the beginning of every major time step. The developed data transfer interface for all operations will run in the update phase, to prevent potential interferences with the Simulink iterative solution and reduce computational time. During this phase, the communication with the Simulink model only happens each major time step.

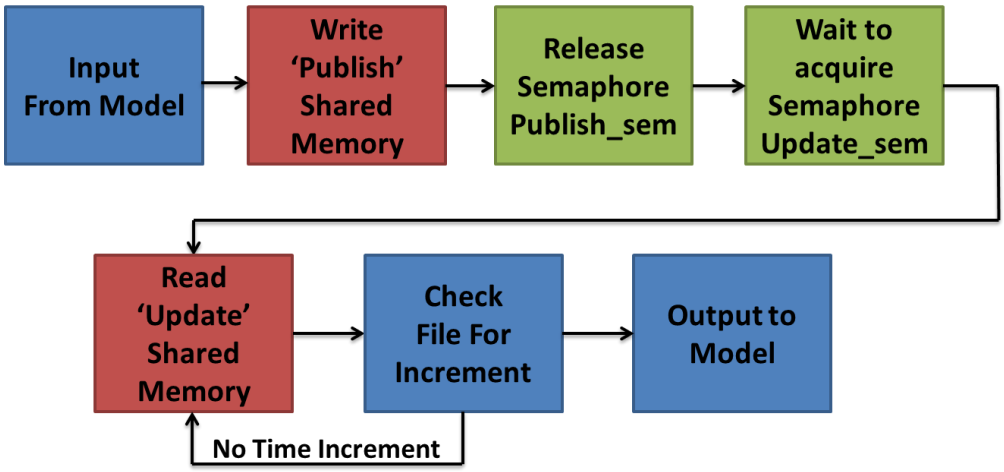

Figure 3.2: Flow diagram of Matlab Simulink S-Function data and control transfer operations.

The flow diagram of the S-function (Figure 3.2) describes the operations in the update phase of the final S-Function design. First, the S-Function collects the input system parameters to the function block from within the Simulink simulation. The function then writes this data to the 'Publish' data output communication lane. Programming semaphores (Semanchuk 2018b) are used to control the access to the stored shared memory. The S-Function releases the 'Publish' semaphore after writing the 'Publish' data, which indicates to the external interface program to begin operations. The S-function then waits for the 'Update' semaphore to indicate completion of the external program's operations before reading the 'Update' data lane. In either case, after the 'Update' data is read and confirmed to be current and complete, the data is sent to the output section of the functional block in the Simulink model (Fig. 3.2).

An external interface program written in Python is developed, to facilitate external data operations (Fig 3.3). Python is a versatile programming platform that utilizes all common

communication protocols. The python interface continuously reads the data published by the S-Function until it detects an updated time value. It then writes that data to the 'Update' data lane (Fig 3.3). The developed python interface waits for the 'Publish' semaphore to indicate that the S-Function has written new data values, before reading the 'Publish' data lane, and makes sure the data is current. It writes this data to the 'Update' data lane before releasing the 'Update' semaphore to indicate to the S-Function that its operation is completed.

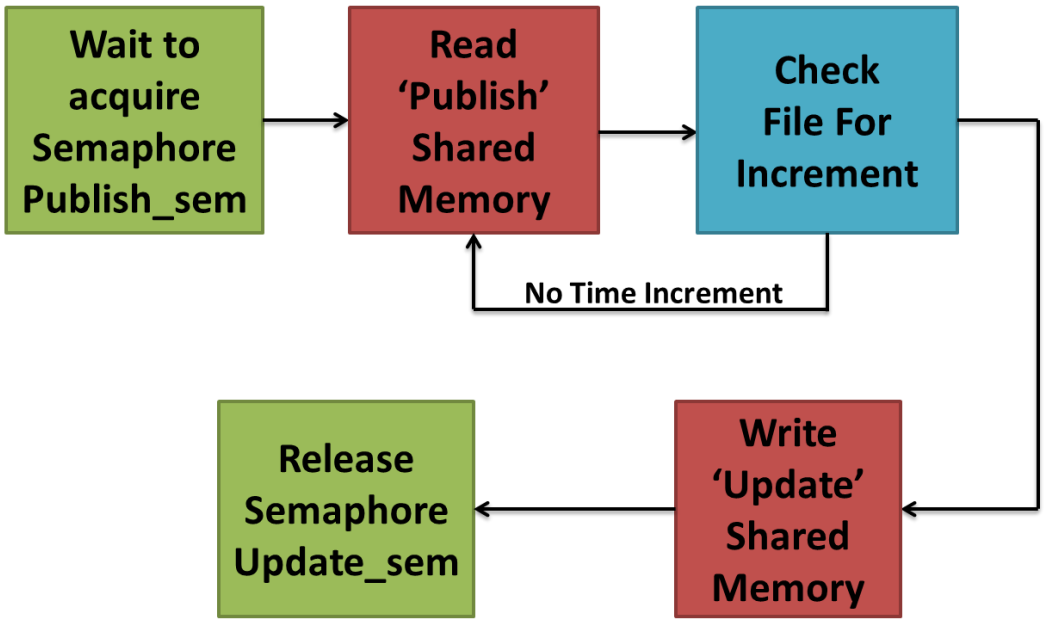

Figure 3.3: Flow diagram of Python external interface data and control transfer operations.

3.2.1 Data Transfer Methods Investigated

This subsection investigates four data transfer methods, namely (a) the text file transfer method, (b) the serial data communication method, (c) the shared memory communication method, and (d) the method of Transmission Control Protocol over Internet Protocol (TCP/IP). The most simplistic method of the text file transfer, utilizes two shared text files that data is written to and read from. The S-Function and the external interface have write permission for only one of the two text files. This ensures no interference between the read and write operations for the 'Update' and 'Publish' text files.

The serial communication method is a standard communication protocol that transmits 8-bit binary data packets between two serial ports (Liechti 2015). To facilitate communication, a third program is needed to generate two virtual serial ports that are interconnected to the S-Function and the external interface. The data is packed into 8-bit binary packets to transmit via the serial

ports. Once received, the data is unpacked back into its original format. One of the major benefits of the serial communication method is inherent synchronicity. Two-way communication cannot happen simultaneously. To receive data, a program 'listening' to the serial communication line waits to receive the entirety of the message before continuing. This ensures synchronization, since after transmitting data the S-Function switches to listening on the serial port. It will hold the process until receiving a response from the external interface. Likewise, after transmitting out the data to the S-Function, the external interface switches to listening and waits until the S-Function responds at the next time step.

The shared memory method is a protocol incorporated into every modern operating system for the purpose of inter-process communication (Semanchuk 2018a). Shared memory allocates a portion of the system memory for data storage, which can be accessed and manipulated by multiple processes. Functionally, this is very similar to the text data transfer method, with some exceptions. Shared memory does not need computationally slow file operations, such as opening and closing files every time they are read or written. Not needing to open and close files makes the shared memory method a far faster and light weight method of data transfer.

The TCP/IP method is a protocol commonly used in computer networking. The data is first converted into binary packets which are then transmitted to the IP address of the intended recipient where they are unpacked into readable data (McMillan 2019). The TCP/IP method is quickly ruled out, because it has unmanageable issues with data packet loss and poor response time with the Simulink simulation, leading to significant control timing mismatch and instabilities in the simulation solution. The other three methods are developed further into functional transfer interfaces and benchmarked against each other to determine which is best for timely, reliable, and synchronous communication and control.

3.2.1.1. Transfer Method Benchmarking Setup

A line diagram of the benchmarking setup used to test the different interface methods is depicted in Figure 3.4. Two Sine waves are generated and multiplexed into a single vector, which is sent to the input of the S-Function. The S-Function then sends this data out to the python interface. The external python interface receives the data and sends it back to the S-Function. After receiving the data from the external python interface, the S-Function outputs this data to the simulation where analyzed to determine the difference between the signal sent and that received. In a subsequent benchmarking setup, the Python interface is linked with an external

programmable logic controller (PLC) (Fig. 3.4). The PLC receives the simulation data from the python interface and returns control decisions. Details of this implementation using the Raspberry Pi 3B+ PLC are given in Section 2.2.1 of this progress report.

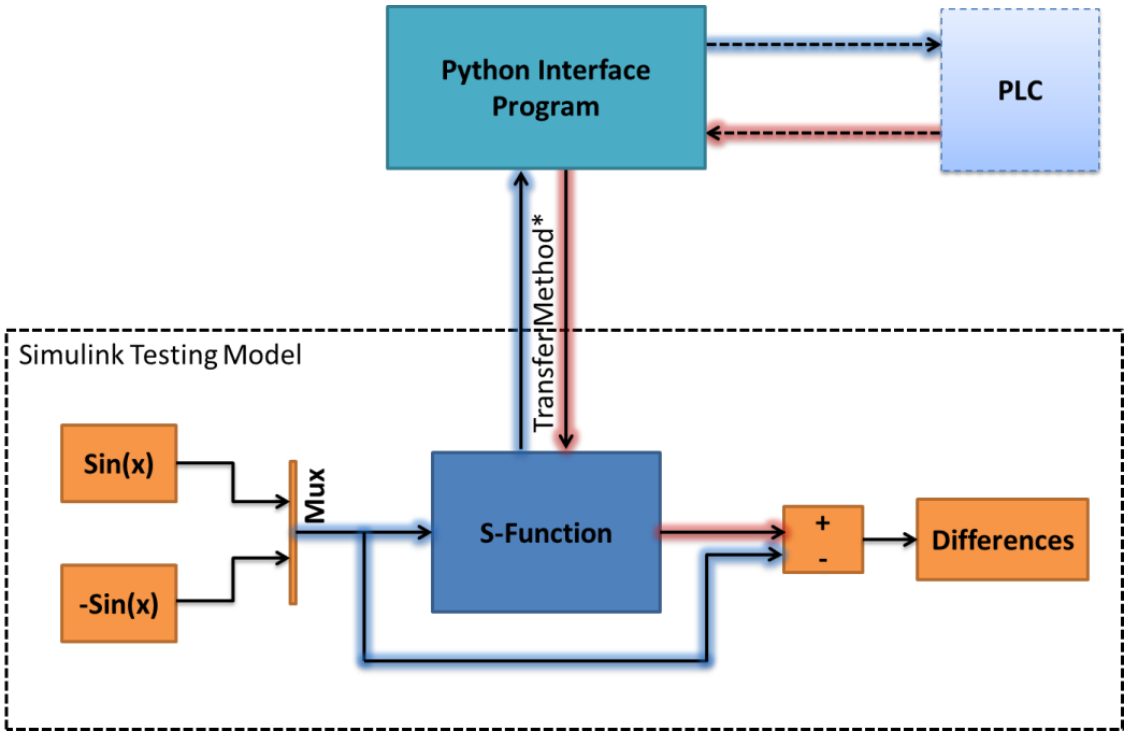

Figure 3.4: Benchmarking setup for data transfer using different interface methods.

3.2.1.2 Results and Discussion

Each of the communication method is tested for 2000 seconds of simulation time using the benchmarking setup in Fig 3.4, and the collected data is compared. The testing results revealed significant issues initially with each of the three transfer methods examined. The benchmark testing of the text file transfer and shared memory methods revealed a mis-synchronization problem resulting in a time shift and significant errors when processes try to read and write the same data simultaneously (Figure 3.5). This situation is referred to as a race condition. The serial communication method does not have a synchronization issue, and the data maintained high fidelity with this method. However, as shown in Figure 3.6, the serial communication method is orders of magnitude slower than text file transfer and the shared memory methods. This increases the simulation time required for each time step to the point where the real run time exceeds the simulated time. This makes simulating the response of PLCs to real time data practically impossible. The serial data transfer method is thus excluded from further consideration, and

focus is shifted to alleviating the synchronization issues with the text data transfer and shared memory methods.

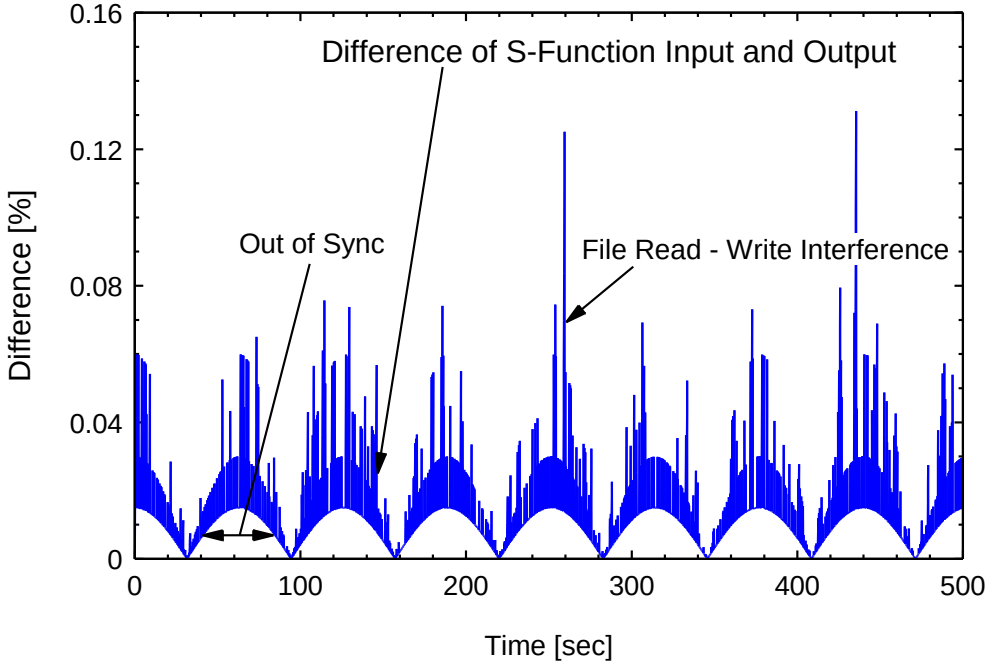

Figure 3.5: Text file and shared memory methods between S-Function input and output.

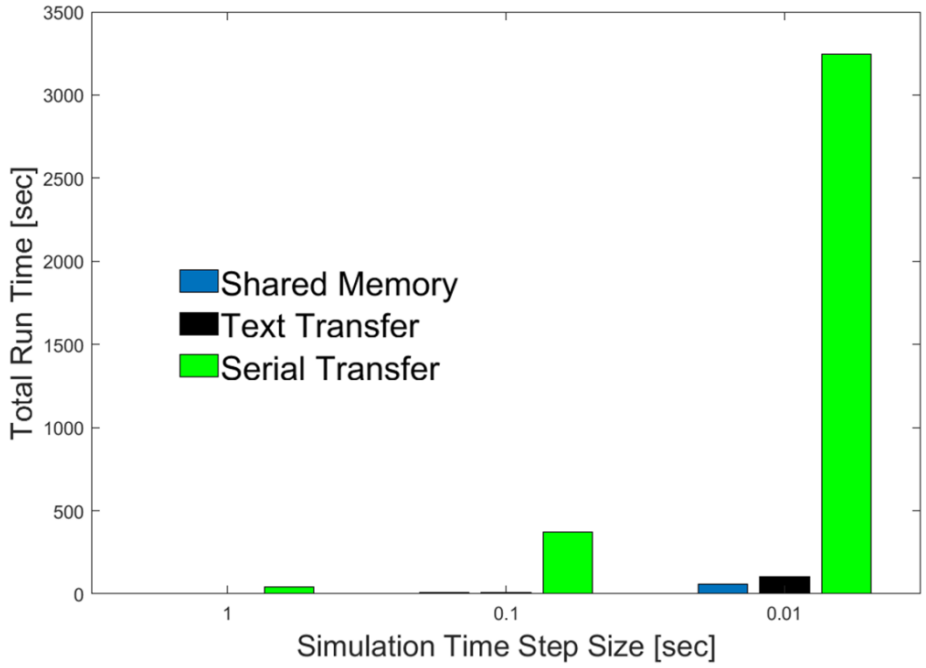

Figure 3.6: Comparison of the total run time of the different data transfer methods examined.

3.2.2 Improved Synchronicity

Semaphores, a part of the inter-process communication protocol (Semanchuk 2018b), are a non-negative integer stored in the shared memory. When a process "releases" semaphore, it is incremented by one (1). When a process is "acquiring" or waiting on a semaphore, it holds operations until it reads that the semaphore is greater than 0. Once a process successfully "acquires" the semaphore, it decrements the integer by 1. Synchronization is enforced using the semaphore protocol by ensuring that each side of the interface waits to access the data while the other side has the semaphore.

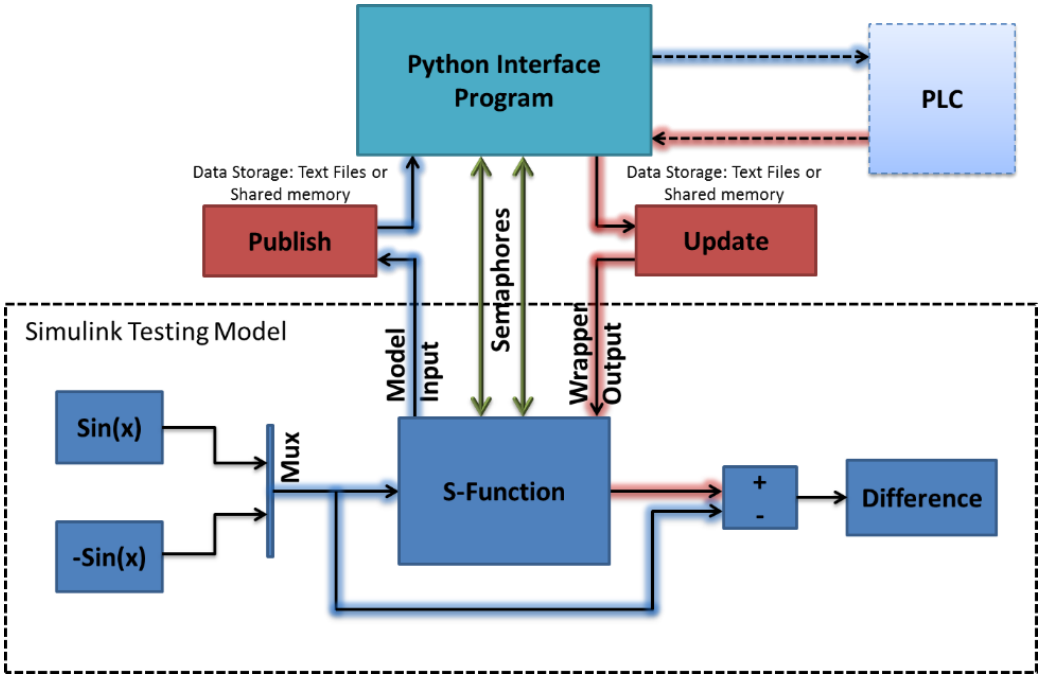

Figure 3.7: Diagram of improved transfer interface setup with semaphores.

Figure 3.7 depicts the setup of the improved transfer interface methods with implemented semaphores to ensure synchronicity. For this application, two named semaphores are used: the "Update" semaphore and the "Publish" semaphore. When the S-Function writes data to the "Publish" file or shared memory space, it releases the "Publish" semaphore. This indicates to the python interface that the S-Function is finished writing the next data set. The python interface then reads the "Publish" file or memory space and performs any operations needed before writing the "Update" file or memory space. Once the python interface writes to the "Update" file or memory space, it releases the "Update" semaphore to indicate to the S-Function that it is done writing the data. The S-Function detects that the semaphore is released and reads the "Update"

file or memory space, which is then output to the model. This process repeats every time step, ensuring that both programs run in perfect lock-step operation. Such synchronization eliminates any time shift and read/write overlap.

3.2.2.1 Results and Discussion of Improved Setup

The recorded difference between the input and output of the S-Function using the interface with semaphores is plotted in Fig 3.8. The significant error reduction due to the implementation of semaphores for synchronization, when compared to the results without semaphores, is clear (Fig 3.5). The little error is due to some round off in the 7^{th} decimal place, compared to a difference of 0.13% without the semaphores.

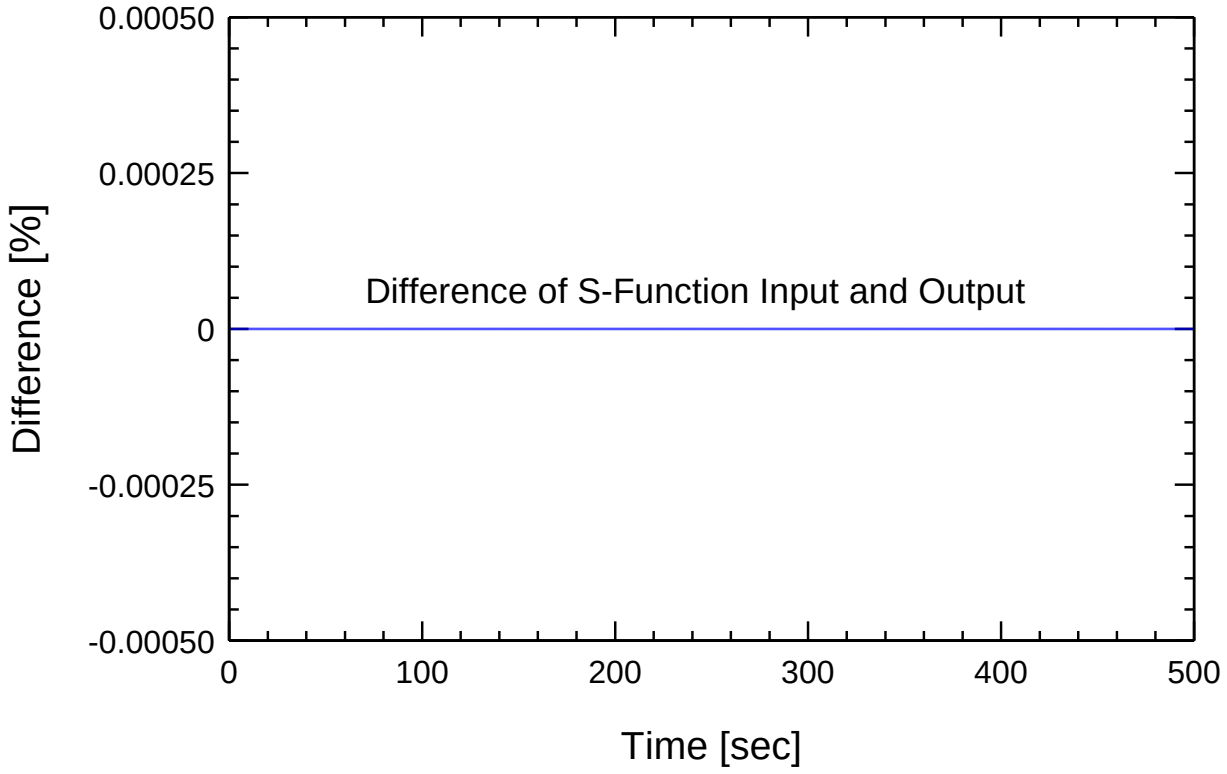

Figure 3.8: Difference of S-Function Input & Output with Improved Setup

The simulation run time comparison in Fig 3.9 shows the speed difference between the text data transfer method and the shared memory data transfer method. While text transfer is a perfectly viable data communication method, it is much more computationally expensive than the shared memory method. Consequently, the shared memory method is selected for integration into the dynamic space nuclear reactor power system model (DynMo-CBC) (El-Genk, Tournier,

and Gallo 2010), and is discussed next.

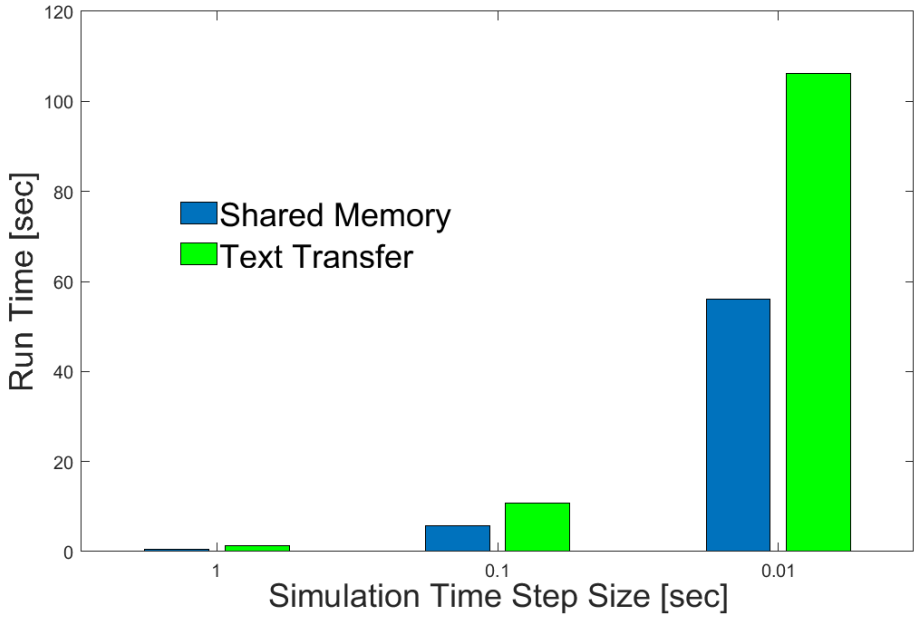

Figure 3.9: Total run time comparison with improved setup using semaphores.

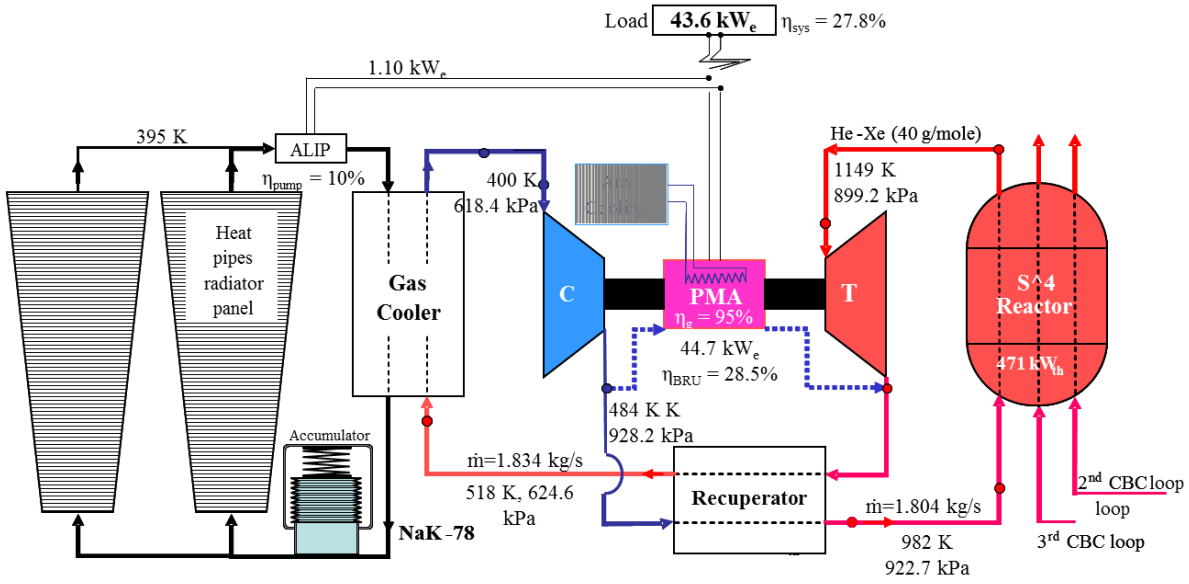

Figure 3.10: A layout of the DynMo-CBC integrated space reactor power system model (El-Genk, Tournier, and Gallo 2010).

3.3. DynMo-CBC Model

DynMo-CBC, a dynamic simulation model for a space nuclear power system with three closed Brayton-Cycle (CBC) energy conversion loops, was developed by the University of New

Mexico's Institute for Space and Nuclear Power Studies (UNM-ISNPS) (El-Genk, Tournier, and Gallo 2010). The modeled integrated power system comprises a Submersion Subcritical Safe Space (S^4) gas cooled reactor (King and El-Genk 2009) and 3 CBC energy conversion loops, each coupled to a separate sector of the nuclear reactor. The three sectors of the core of the S^4 reactor are neutronically and thermally coupled, but hydraulically independent, and each serviced by one of the three CBC loops. This space power system is designed for space exploration missions or planetary surface power is designed for the avoidance of single point failures (El-Genk, Tournier, and Gallo 2010).

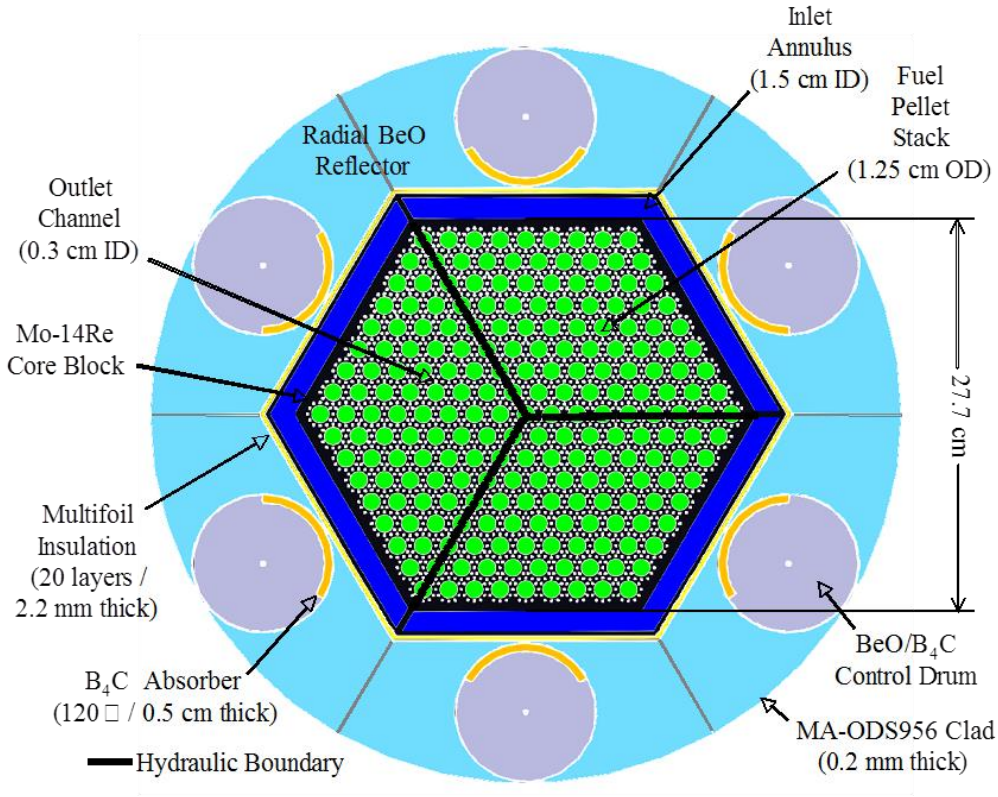

Figure 3.11: Radial cross section of the DynMo-CBC S^4 reactor core (El-Genk, Tournier, and Gallo 2010; King and El-Genk 2009).

Figure 3.10 depicts one of the three CBC loops in the S^4 CBC power system. Each CBC loop has a single shaft turbomachine, with a centrifugal turbine and compressor and a permanent magnet alternator (PMA). The waste heat is rejected into space by water heat pipes radiator panels that are thermally coupled to the CBC He-Xe gas loop by a secondary NaK-78 liquid metal loop. Fig 3.11 presents a section view of the S^4 reactor (King and El-Genk 2009). It is controlled using six rotating B_4C/BeO segmented control drums within the BeO radial reflector.

The S^4 CBC power system normally operates at a reactor thermal power of 471 kW_{th} and delivers 130.8 kW_e of load power for up to 12.4 years of full power operation. The fully integrated transient DynMo-CBC space reactor power system model is implemented within the Simulink platform (Fig. 3.10).

The PLC and transfer interface are linked to the DynMo-CBC model to only control the rotation of the control drums of the S^4 reactor (Fig 3.11). Once successfully implemented, more complex control systems could be easily introduced in the PLC programming. The python interface bridges the communication gap between the external PLC and the Simulink DynMo-CBC model. The response of the DynMo-CBC model linked to the external PLC is benchmarked against the results of the DynMo-CBC model, using internal control logic built within the Simulink model.

3.3.1 DynMo-CBC External Controller Integration

Figure 3.12 is a diagram of the improved interface integrated into the DynMo-CBC model for controlling the rotation of the control drums (Fig. 3.11) with a Raspberry Pi PLC. The Simulink based DynMo-CBC model outputs state variables to the S-Function, which writes the data to the 'Publish' shared memory space and releases the 'Publish' semaphore. The python interface detects the release of the 'Publish' semaphore and reads the 'Publish shared memory space. The interface then converts the read data to a Modbus compatible format and sends it via Modbus to the Raspberry Pi PLC. The PLC reads the received data from the python interface and determines the required angular rotation rate of the control drums (Fig. 3.11). The PLC communicates the drums' actuation speed via Modbus to the python interface, which writes it to the 'Update' shared memory space. The 'Update' semaphore is then released by the Python interface. The S-Function detects the release of the 'Update' semaphore and continues to read the data in the 'Update' shared memory space. The drums' rotation rate data is then output from S-Function into the simulation, and acted on the drum control commands. The drum controller for both the internal control and the external PLC operate as follows:

a) The controller checks the simulation time,
b) If the time is greater than 60 seconds, drum rotation begins,
c) The rotation angle of the drums is measured, and if below 52° rotation continues,
d) If above conditions are true, the rotation rate of the control drumms is set to 0.0025°/sec.

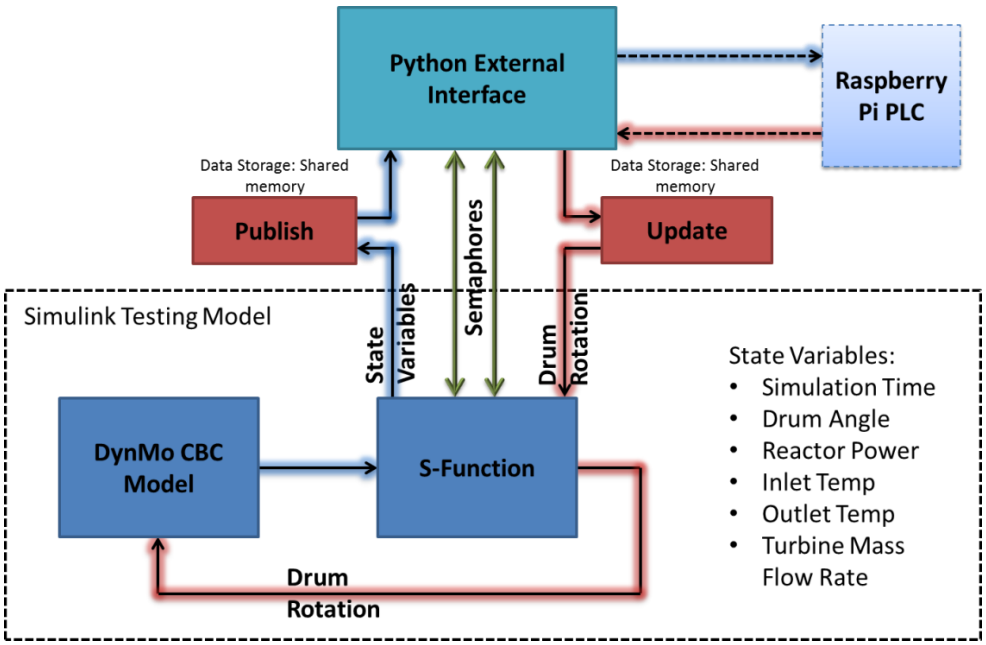

Figure 3.12: Diagram of improved data and control transfer interface to DynMo-CBC

The rotation of the control drums outward increases the external reactivity for starting up the S^4 reactor (El-Genk, Tournier, and Gallo 2010). Figs. 3.13-3.15 show the results of a simulation of the startup of the S^4-CBC power system. At a time, t = 60 s, the PLC commands the control drums to rotate outward, inserting external reactivity into the reactor core (Fig. 3.13). This increases the reactor thermal power, causing the reactor core and coolant temperatures to increase (Fig. 3.14). The rise in the core temperatures results in a negative temperature reactivity feedback that temporarily decreases the total reactivity and subsequently the reactor thermal power, causing the first spike seen in Fig. 3.14. As the external reactivity insertion continues, by further rotating the control drums continues, the temperature feedback due to the increase in reactor power, equals the external reactivity insertion, resulting in a steady increase in the reactor power and core temperatures until the end of the reactivity insertion during the reactor startup. This is when the control drums reach a position of 52° rotated outward (Fig. 3.14). Initially during the reactor startup transient, a battery is used to supply power to a motor rotating the single shaft turbomachinery units until reaching a rotation rate of 45,000 rpm at which net shaft-power becomes zero and the battery disconnected (Fig. 3.15). As the reactor thermal power increases during startup, the power generated by the turbine exceeds that consumed by the compressor, and the power generation becomes net positive. After the system reaches steady state, each CBC unit supplies 43.6 kW_e.

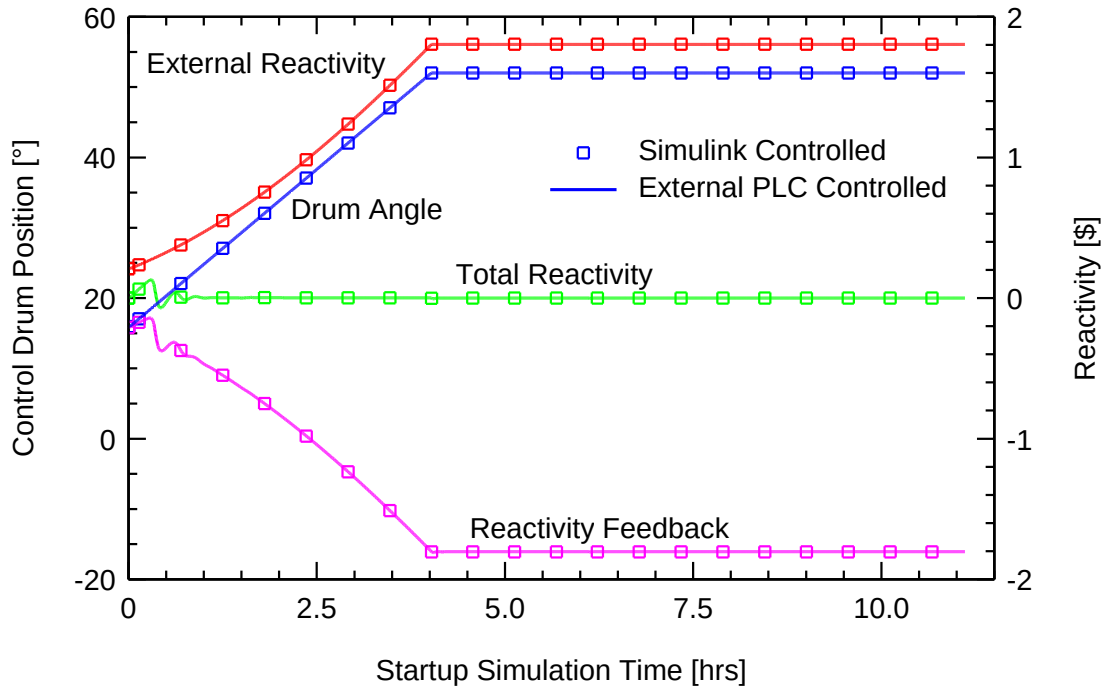

Figure 3.13: Core reactivity and control drum position during reactor startup.

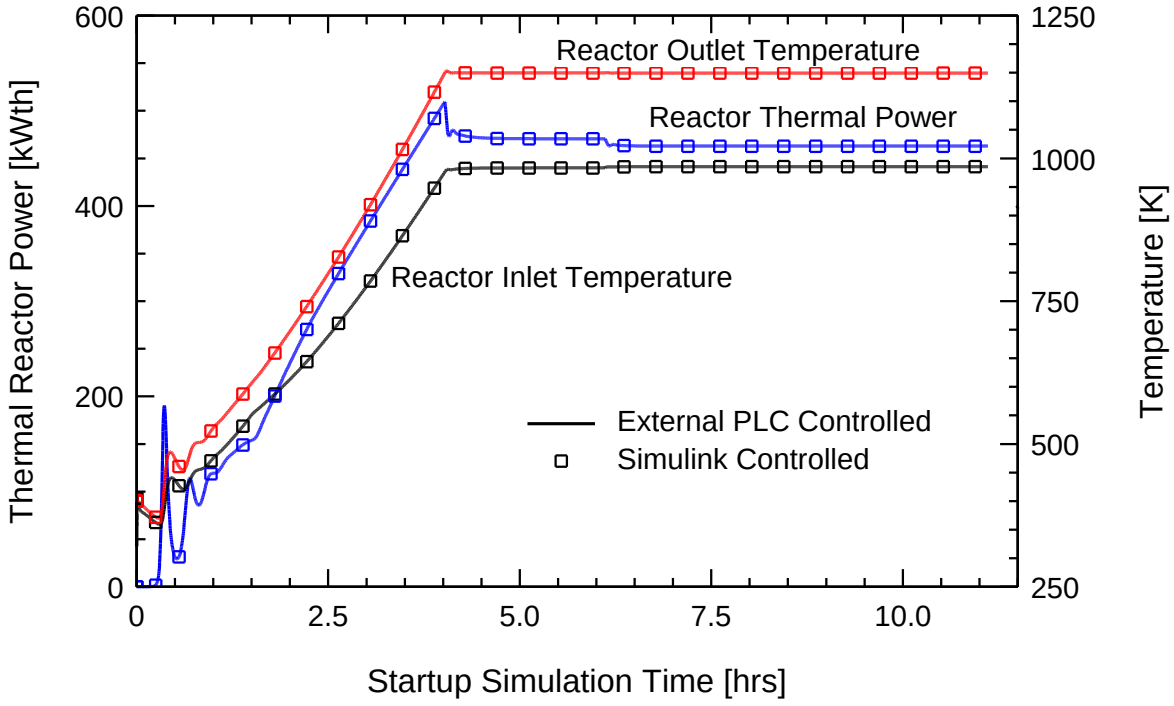

Figure 3.14: Reactor thermal power and reactor inlet and outlet temperatures during reactor startup.

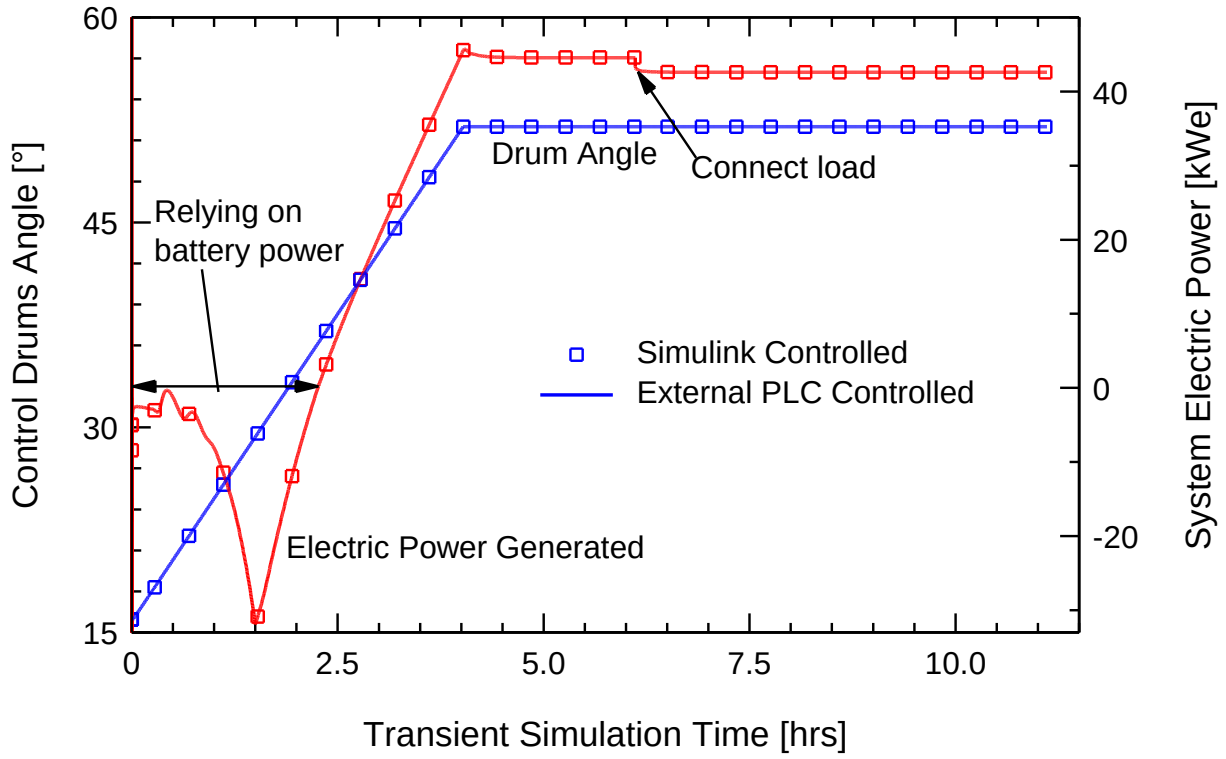

Figure 3.15: System electric power generation and control drum position during startup.

In Figs. 3.13-3.15 the solid lines for the system state variables generated by the DynMo-CBC model connected to the external Raspberry Pi PLC through the developed interface. The square symbols in these figures are the values produced by the DynMo-CBC simulation using internal control logic of the control drums' rotation (Figs. 3.13-3.15). The results show no difference between the state variable values calculated during the startup transient by the DynMo-CBC model controlled by the PLC and those calculated with an internal Simulink control model. Figs. 3.13-3.15 show that the complex feedback behavior captured by DynMo-CBC is not impacted by the external PLC as a part of the power system startup transient. This test successfully demonstrates the capability of the interface program to enable an external PLC to control a dynamic simulation model of a space nuclear power system running in Matlab Simulink.

3.4. Summary

The work presented in Section 3.0 investigated four transfer methods for communication between Matlab Simulink and an external PLC. These are (a) the text file transfer method, (b) the serial communication method, (c) the shared memory method, and (d) the TCP/IP method. The TCP/IP method was excluded due to poor performance. The serial communication method is too

slow to be effective. The text file transfer and shared memory methods initially tested as fast enough, but have issues with synchronization. However, this is remedied with semaphores for enforcing synchronicity. After synchronous implementations, the shared memory method is much faster than text file transfer method, and is selected as the best transfer method for the interface with Simulink.

The shared memory data and control interface is successfully implemented in the DynMo-CBC Simulink model of a fully integrated space nuclear reactor power system for controlling the rotation of the control drums rotation using a Raspberry Pi PLC. The data of the DynMo-CBC model state variables with the drums being controlled by the Raspberry Pi PLC and that of the power system model with drums controlled by logic implemented internally within Simulink are compared. The results with the Raspberry Pi PLC control are identical to those of the internal Simulink controller, thus demonstrating viability of the data and control interface for use in complex dynamic systems.

4. Summary and Conclusions

This report detailed the work performed by UNM-ISNPS in conjunction with SNL on the implementation and validation of PLC emulation and data transfer. This effort is part of the Nuclear Instrumentation & Control Simulation (NICSim) project funded by a DOE NEUP award in 2018. The work presented in this report successfully develops and validates an emulation methodology for modeling the PLCs for use in the safety I&C system of a nuclear reactor power plant. The implemented methodology links a PLC to a transient model of a fully integrated nuclear reactor power plant using a fast-running and robust interface.

The PLC emulation methodology identifies the characteristic digital and physical signatures of the physical PLC. The PLC emulation is validated by ensuring that the behavior of the emulated PLC sufficiently matches that of the real hardware. The validation effort characterizes the digital signatures of the network response and that of recorded network traffic between the PLC and server PC running the Simulink process simulation. Although the emulated PLC has a higher network bandwidth capacity than the real PLC, this does not significant impact the communication speeds. Setting the major simulation time step > ~50 ms results in no significant difference in the network response time of the real and emulated PLCs. The PCAP analyses shows that the emulated PLC generates similar variety and relative proportions of network traffic packets as the real PLC.

The signature of the validation physical characterizes the actuation response time and the sampling rate for the real PLC and compares them to those of the emulated PLC. Results show that when running the Simulink simulation with a major time step ≥ 200 ms, the actuation signal response times of the real and emulated PLC agree to within < 2%. Testing the sampling rate of the OpenPLC programming shows that, with a sampling rate ≥ 50 ms, both the real and emulated PLCs agree with the specified sampling time set in OpenPLC. The results show the emulated PLC has actuation responses and sampling rates essentially indistinguishable from those of the real PLC. From a cybersecurity perspective, this PLC emulation runs the same software, communicates using the same communication protocols, and generates the same types and proportions of network traffic data types as the real device.

This research also successfully develops an interface linking an emulated or real PLC to a dynamic simulation model of a space nuclear reactor power system within Matlab Simulink. This interface transmits the simulation state variables of the plant condition from the Simulink model

to the PLC and transmits back the generated control signals to actuate a system within the simulation model. This interface is fast, reliable, and synchronous. It is compiled by the Matlab Simulink C coder to produce a stand-alone executable of the nuclear reactor power system simulation model. This effort investigated four different methods of inter-process communication to transmit data values between a physics-based Matlab Simulink simulation model and a python script. The investigated methods include: (a) the text file transfer method, (b) the serial communication method, (c) the shared memory method, and (d) the TCP/IP method.

Implementing these methods within Matlab Simulink requires developing special Matlab S-Functions written in the C programming language. Testing of the four data transfer communication methods show that shared memory communication method and the text file transfer method are the fastest and most reliable. The serial communication method is much slower than the text file transfer and shared memory methods. The TCP/IP method communication between Matlab and the python script is unreliable, causing instabilities in the transient Simulink model. The shared memory method, the fastest by far, is selected for the interface, using semaphores to ensure synchronicity with the physics-based Simulink model.

The effectiveness of this interface is demonstrated by linking a physical Raspberry Pi based PLC, running ladder logic control programming using OpenPLC, to a Matlab Simulink dynamic model of an integrated space nuclear reactor power system. The PLC controls the rotation of the reactor's control drums. The connected PLC successfully commanded the simulated space nuclear reactor to start up to full power, nominal steady-state operating conditions. The transient behavior of the power system controlled by the PLC is indistinguishable to that of a setup of internal control logic built within Simulink.

5. Application to NICSim Project

The research detailed in this progress report developed and demonstrated important elements to be implemented in the NiCSim platform, currently being developed by UNM-ISNPS in conjunction with SNL, under a 2018 DOE NEUP award. Fig. 5.1 shows a line diagram of the different components of the NICSim platform. The validated PLC emulation described in Section 2 will be integrated within the SCEPTRE framework to represent the different PLCs within the safety I&C system architecture of a commercial nuclear reactor power plant. Integrating the emulated PLC in the SCEPTRE framework at SNL is currently underway, and nears completion. The emulated PLC developed here will be adapted to represent the different PLCs within the nuclear power plant's protection and safety monitoring I&C system, by writing ladder logic control programs to run within OpenPLC on the emulated systems.

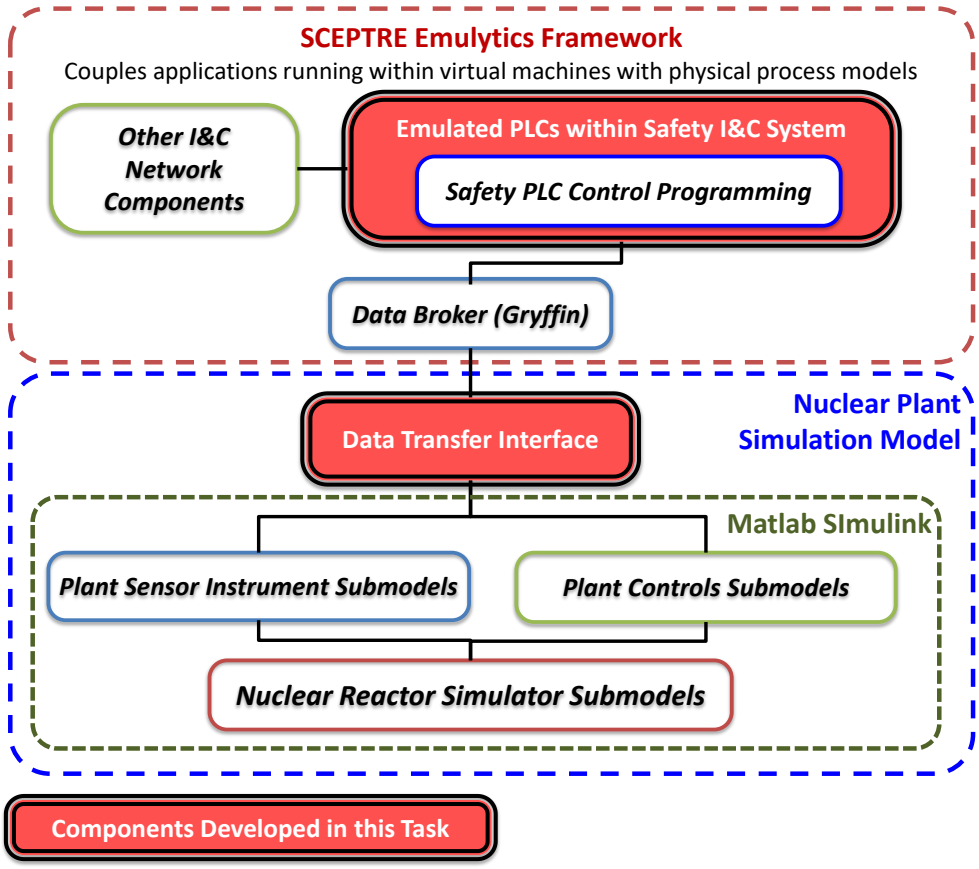

Figure 5.1: Line diagram of the elements within the NICSim platform for the implementation of the physics-based models of a nuclear reactor power plant to the SCEPTRE emulytics framework.

The selected data transfer interface, developed and tested in Section 3, is successfully

demonstrated to effectively link a PLC to a dynamic simulation model of a space nuclear reactor power system. Future application of this interface, for facilitating the communication and control between a Simulink physics-based model of a commercial nuclear plant and the emulated PLCs and virtual network, will be implemented within the SCEPTRE framework (Fig. 5.2). To accomplish this linkage, a python wrapper of the interface will communicate with a data broker program within the SCEPTRE framework, called Gryffin. Gryffin handles the direct communication with the virtual machines, representing the PLCs, and other I&C network components (Fig. 5.1). The component submodels representing the primary loop of a Pressurized Water Reactor (PWR) and plant instrument sensor and controls submodels (Fig. 5.1) will send and receive values to and from the PLCs through this interface.

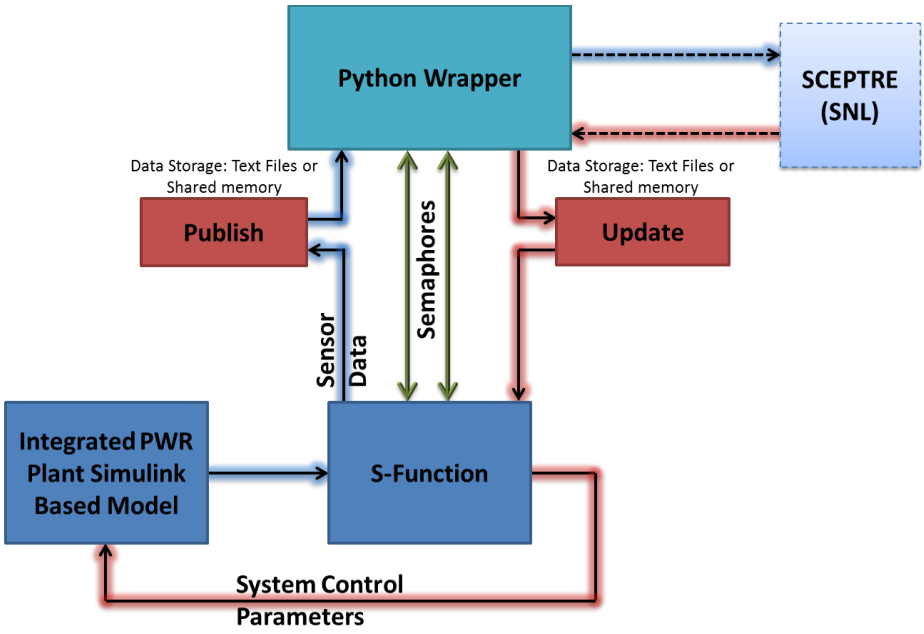

Figure 5.2: Line diagram showing developed interface to connect the nuclear plant simulation model to SCEPTRE within the NICSim platform.

The validated PLC emulation and developed interface program will enable future simulation of different operation and safety I&C system architectures linked to the dynamic, physics-based model a nuclear power plant. Future implementation and successful completion of this work include investigating the physical impacts of targeted cyber-attacks on plant I&C system architectures, aiding in training plant operators in identifying signs of a cyber-attack, and helping develop metrics to quantify how a cyber-attack propagates throughout nuclear power plant I&C system networks.

6. References

Alves, T.R., Buratto, MM, Mauricio de Souza, F., and Rodrigues, T.V., 2014. "OpenPLC: An open source alternative to automation," in proceedings IEEE Global Humanitarian Technology Conference (GHTC 2014), San Jose, CA, USA, DOI: 10.1109/GHTC.2014.6970342

Alves, T., 2018, OpenPLC 1.0, https://www.openplcproject.com/.

Chunjie, Z. and Hui C., 2009. "Development of a PLC Virtual Machine orienting IEC 61131-3 Standard," in proceedings 2009 International Conference on Measuring Technology and Mechatronics Automation, IEEE Computer Society, DOI 10.1109/ICMTMA.2009.422

Collins, Galen, 2019. pymodbus 2.2.0, https://github.com/riptideio/pymodbus/.

Dragos, Inc., 2017. CRASHOVERRIDE, Analysis of the Threat to Electric Grid Operations version 2.20170613, www.DRAGOS.com.

El-Genk, M.S., Tournier, J-M., Gallo, B.M., 2010. Dynamic Simulation of a Space Reactor System with Closed Brayton Cycle Loops, Journal of Propulsion and Power, 26(3), 394-406.

Gasser, T., 2013. Virtual Machine and Code Generator for PLC-Systems, Thesis submitted to the University of East London for the degree of Master of Philosophy

Karnouskos, S., 2011. "Stuxnet Worm Impact on Industrial Cyber-Physical System Security," in proceedings IECON 2011 - 37th Annual Conference of the IEEE Industrial Electronics Society, Melbourne, VIC, Australia, 7-10 Nov, 2011, DOI: 10.1109/IECON.2011.6120048.

Kerahroudi, S .K., Alamuti, M. M., Li, F., Taylor, G. A., Bradley, M. E., 2014, Application and Requirement of DIgSILENT PowerFactory to MATLAB/Simulink Interface, PowerFactory Applications for Power System Analysis, 297-322.

King, J.C., and El-Genk, M.S., 2009. Thermal-Hydraulic and Neutronic Analyses of the Submersion-Subcritical, Safe Space S^4 Reactor, Nuclear Engineering and Design, 239, 2809-2819.

Korsah, K., et al., 2008. Instrumentation and Controls in Nuclear Power Plants: An Emerging Technologies Update, US NRC Technical Report NUREG/CR-6992, Washington, DC.

McMillan, G., 2019. Socket Programming HOWTO, https://docs.python.org/3/howto/sockets.html.

National Research Council, 1997. Digital Instrumentation and Control Systems in Nuclear Power Plants, Safety and Reliability Issues, Final Report, National Academy Press, Washington, D.C.

Perlroth, N., 2019. “Hackers are Trageting Nuclear Facilities, Homeland Security Dept. and F.B.I. Say”, New York Times, June 19, 2019.

Sandia National Laboratories, 2016. SCEPTRE, Sandia Document SAND2016-8095C.

Semanchuk, P., 2018a. POSIX IPC for Python, http://semanchuk.com/philip/posix_ipc/.

Semanchuk, P., 2018b. System V IPC for Python, http://semanchuk.com/philip/sysv_ipc/.

Siemens, 2018. Realization of a SIMIT Shared Memory Coupling with Matlab, https://support.industry.siemens.com/cs/document/109761656/realization-of-a-simit-shared-memory-coupling-with-Matlab?dti=0&lc=en-HU.

Thamrin, N.M. and Ismail M.M., 2011. “Development of Virtual Machine for Programmable Logic Controller (PLC) by Using STEPS™ Programming Method,” in proceedings 2011 IEEE International Conference on System Engineering and Technology (ICSET), IEEE.
The MathWorks, Inc., 2018. Simulink Version 9.2 (R2018b).

The MathWorks, Inc., 2019. C Source MEX Files – MATLAB & Simulink, https://www.MathWorks.com/help/Matlab/write-cc-mex-files.html.

VMware, 2019. VMware Workstation 15 Pro.

Wireshark, 2019. Wireshark User’s Guide Version 3.1.0, https://www.wireshark.org/docs/wsug_html_chunked/, accessed July 2019.