IMPERIAL

Cloud-based Co-simulation

Author
Yunzhong Gu
CID: 02256233

Supervised by
Dr Yunjie Gu

A Thesis submitted in fulfillment of requirements for the degree of
Master of Science in Control and Optimisation

Department of Electrical and Electronic Engineering
Imperial College London
2024

Abstract

Simulation is a crucial tool for validating the dynamic stability of power systems. The increasing integration of inverter-based resources brings new challenges, primarily due to the proprietary control algorithms that define inverter models. This project proposes a cloud-based co-simulation platform to address these challenges. In this platform, inverter models and power system models are simulated on separate computers, owned by the inverter vendors and system operators, respectively. These computers exchange signals over the cloud via designed model communication interfaces, simulating the interactions between inverters and power systems. This approach facilitates joint simulation among multiple parties without disclosing proprietary models. Additionally, the project tests the co-simulation of large-scale multi-bus power system models on this platform and utilizes communication interfaces to combine co-simulation with parallel simulation, further accelerating the model simulation speed. The project introduces the design and implementation process of the co-simulation and conducts specific tests and analyses to compare the transient simulation results of power systems, demonstrating the accuracy and feasibility of the co-simulation.

Keywords - Co-Simulation, Real-Time Simulation, Communication Interface, Parallel Simulation, Power System Simulation, Power System Transients

Declaration of Originality

I hereby declare that the work presented in this thesis is my own unless otherwise stated. To the best of my knowledge the work is original and ideas developed in collaboration with others have been appropriately referenced.

Copyright Declaration

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Dr. Yunjie Gu, for his invaluable guidance and insightful advice throughout the duration of my project.

I am also thankful to my colleagues at Imperial College London for fostering an inspiring and dynamic research environment.

Lastly, I am deeply grateful to my parents for their continuous encouragement and unwavering support.

Contents

Abstract i

Declaration of Originality iii

Copyright Declaration v

Acknowledgments vii

List of Figures xi

1 Introduction 1

1.1 Co-Simulation Framework . . . 1

1.1.1 Decoupling Method and Interface Design . . . 3

1.1.2 Model Decoupling . . . 5

1.1.3 Communications Protocol . . . 5

1.2 Parallel Simulation . . . 8

1.2.1 Factors affecting simulation speed . . . 8

1.2.2 Implementing Parallel Simulation in Matlab . . . 8

1.2.3 Parallel Simulation Planning . . . 9

1.3 Scope of the Thesis . . . 11

2 Co-simulation of grid-connected inverter systems 13

2.1 Grid-Connected Inverter Model and Partitioning Strategy . . . 13

2.1.1 Original Model Introduction . . . 13

2.2 Simulation Process . . . 15

2.2.1 Model Setup and Interface Compilation Ideas . . . 15

2.2.2 Network Connection . . . 17

2.2.3 Simulation Workstation . . . 17

2.3 Simulation Results . . . 19

2.3.1 Co-simulation Results in LAN . . . 19

2.3.2 Co-simulation Results Across Campus Network . . . 22

3 Co-simulation of multi-bus transmission line systems **29**

3.1 116 Bus Transmission Line Model and Partitioning Strategy 29

3.1.1 Original Model Introduction 29

3.2 Simulation Process 32

3.2.1 Interface Communication between Sub-models 32

3.2.2 Parallel Simulation Interface 33

3.2.3 Sub-model Initialization and Startup 33

3.2.4 Simulation Workstation 33

3.3 Simulation Results 34

3.3.1 Co-simulation Results in LAN 34

3.3.2 Co-simulation Results Across Campus Network 37

4 Simulation Platform **43**

4.0.1 Introduction 43

Conclusions **47**

4.1 Conclusions 47

4.2 Future Work 48

A Title of the Appendix **49**

B Title of the Appendix **55**

C Title of the Appendix **61**

D Title of the Appendix **67**

E Title of the Appendix **75**

F Title of the Appendix **79**

Bibliography **93**

List of Figures

1.1 Project Framework. 2

1.2 One-dimensional TLM element with characteristic impedance Z_c and time delay T.[11] . . 3

1.3 Transmission Line Delay Algorithm. 4

1.4 Simulink Model Decoupling. [15] 5

1.5 Timestamps for data alignment. 7

1.6 Difference between fixed and variable step size. 8

1.7 Difference between parsim and batchsim workflow. 9

1.8 Parallel simulation framework. 10

1.9 CPU utilization monitoring. 11

2.1 Original Grid-Connected Inverter model. 14

2.2 Enter Caption 14

2.3 Submodels communication interfaces. 15

2.4 Three-way handshake process using the fopen function. 16

2.5 Network connection between apartment and the Imperial College Campus 17

2.6 Simulation Workstation. 18

2.7 Diagnostic Viewer 18

2.8 Comparison of inverter voltage, grid voltage and synchronization results between the original model and the co-simulation model. 19

2.9 Comparison of inverter current and grid current between the original model and the co-simulation model. 20

2.10 Comparison of Inverter and Grid Power Dynamics in Original and Co-Simulation Models 20

2.11 Comparison of original model and co-simulation model of grid three-phase current (One of the three phases). 21

2.12 Comparison of original model and co-simulation model of inverter active power dynamic. . 22

2.13 Package losses during simulation. 23

2.14 Comparison of inverter voltage and grid voltage between the original model and the co-simulation model (Across network, p.u.). 24

2.15 Comparison of inverter current and grid current between the original model and the co-simulation model (Across network, p.u.). 24

2.16 Comparison of inverter power and grid power between the original model and the co-simulation model (Across network, p.u.). 25

2.17 Comparison of original model and co-simulation model of one phase of inverter three-phase current (Across network, p.u.). . . . 26

2.18 Comparison of original model and co-simulation model of inverter active power dynamic (Across network, V.). . . . 26

2.19 Comparison of inverter power dynamics (p.u.) within and across network conditions. . . . 27

3.1 116 bus transmission line system (original simulation model). . . . 30

3.2 Subsystem expansion within the model. . . . 31

3.3 Synchronous motor terminal voltage (p.u.). . . . 31

3.4 Load three-phase voltage (p.u.). . . . 31

3.5 Interface communication between sub-models. . . . 32

3.6 Simulation models running in parallel pools. . . . 33

3.7 Simulation workstation construction. . . . 34

3.8 Simulation results of motor speed, synchronous motor terminal voltage (LAN). . . . 35

3.9 Simulation results of Load three-phase voltage signal (LAN). . . . 35

3.10 Comparison of motor speed curves between the original simulation model and the co-simulation model (LAN). . . . 36

3.11 Comparison of synchronous motor terminal voltage curves between the original simulation model and the co-simulation model (LAN). . . . 36

3.12 Comparison of load three-phase voltage (single-phase) curves between the original simulation model and the co-simulation model in a local area network (LAN) environment. . . . 37

3.13 Simulation results of motor speed, synchronous motor terminal voltage (Across Network). 38

3.14 Simulation results of Load three-phase voltage signal (Across Netowrk). . . . 38

3.15 Comparison of motor speed simulation results between original model and cross-network co-simulation. (Top: Comparison of simulation curves; Bottom: Error values at each step) 39

3.16 Comparison of synchronous motor end-voltage simulation results between original model and cross-network co-simulation. (Top: Comparison of simulation curves; Bottom: Error values at each step) . . . 39

3.17 Comparison of load three-phase voltage simulation results between original model and cross-network co-simulation. (Top: Comparison of simulation curves; Bottom: Error values at each step) . . . 40

3.18 Comparison of simulation time for different modes. . . . 41

4.1 Simulation platform. . . . 44

4.2 Exporting results from the simulation platform. . . . 45

B.1 Active power. . . . 56

B.2 Reactive power. . . . 56

B.3 Inverter three-phase current A. . . . 57

B.4 Inverter three-phase current B. . . . 57

B.5 Inverter three-phase current C. 58
B.6 Grid three-phase current A. 58

B.7 Grid three-phase current B. 59
B.8 Grid three-phase current C. 59
B.9 Synchronous voltage. 59

C.1 Active power comparison (across networks). 62
C.2 Inactive power comparison (across networks). 62
C.3 Comparison of three phase inverter current a (across networks). 63
C.4 Comparison of three phase inverter current b (across networks). 63
C.5 Comparison of three phase inverter current c (across networks). 64
C.6 Comparison of three phase grid current a (across networks). 64
C.7 Comparison of three phase grid current b (across networks). 65
C.8 Comparison of three phase grid current c (across networks). 65

E.1 Load three-phase voltage (b) error analysis. 76
E.2 Load three-phase voltage (c) error analysis. 76
E.3 Errors in motor 2 motor speed. 77
E.4 Error in motor 2 synchronous motor end voltage speed 77
E.5 Errors in motor 3 motor speed. 78
E.6 Error in motor 3 synchronous motor end voltage speed 78

1 Introduction

Contents

1.1 Co-Simulation Framework **1**
1.1.1 Decoupling Method and Interface Design 3
1.1.2 Model Decoupling 5
1.1.3 Communications Protocol 5
1.2 Parallel Simulation **8**
1.2.1 Factors affecting simulation speed 8
1.2.2 Implementing Parallel Simulation in Matlab 8
1.2.3 Parallel Simulation Planning 9
1.3 Scope of the Thesis **11**

1.1 Co-Simulation Framework

Electrical co-simulation is a technology that integrates different electrical simulation tools and models in a co-simulation environment to simulate and analyze the performance and behavior of complex electrical systems. As the complexity of electrical system design continues to increase, such as with the development of electric vehicles, smart grids, and complex electronic devices, the demand for efficient and accurate simulation becomes increasingly urgent. Traditional co-simulation usually relies on Functional Mock-up Units (FMUs) to achieve model transfer and tool interoperability [1]. However, in the field of electrical co-simulation, the confidentiality and security of models have always been critical issues. Traditionally, compiling simulation models into static or dynamic libraries can protect the models to a certain extent, but some security risks, such as decompilation and unauthorized access, still exist.

Therefore, in this work, a web-based co-simulation approach is proposed that completely avoids the direct provision of the model and carries out data interaction through web interfaces. The core idea of

1

this approach is to create dedicated interface blocks at each end of the participating simulation, through which the data generated during the simulation process is transmitted instead of the model itself. In this way, we can not only effectively protect the intellectual property of simulation models but also leverage distributed simulation and parallel simulation concepts to further accelerate the simulation speed of large-scale power systems, providing strong support for the design and optimization of complex electrical systems [2].

The simulation platform for this project is built on MATLAB/Simulink. MATLAB/Simulink is a commonly used software for system-level simulation, widely applied in steady-state and transient analysis of power systems. When combined with Simscape, MATLAB/Simulink not only performs traditional circuit simulation but also simulates the thermal and electromagnetic characteristics of systems. This combination provides powerful tool support for comprehensive simulation of complex power systems. Additionally, the Parallel Computing Toolbox allows MATLAB to perform parallel simulations, which helps enhance the speed and efficiency of large-scale simulations [3]–[5].

The framework of the simulation system is shown in Figure 1.1. By using a physical model decoupling tool, the original simulation model is divided into four independent sub-models, which run on two separate computers. The computers exchange data through the Co-simulation Interface (Co-SI), which adheres to transmission protocols and ensures the reliability and efficiency of data transfer through handshake rules. Additionally, the sub-models on each computer communicate with each other via the Parallel Interface (PI), further enhancing the simulation speed.

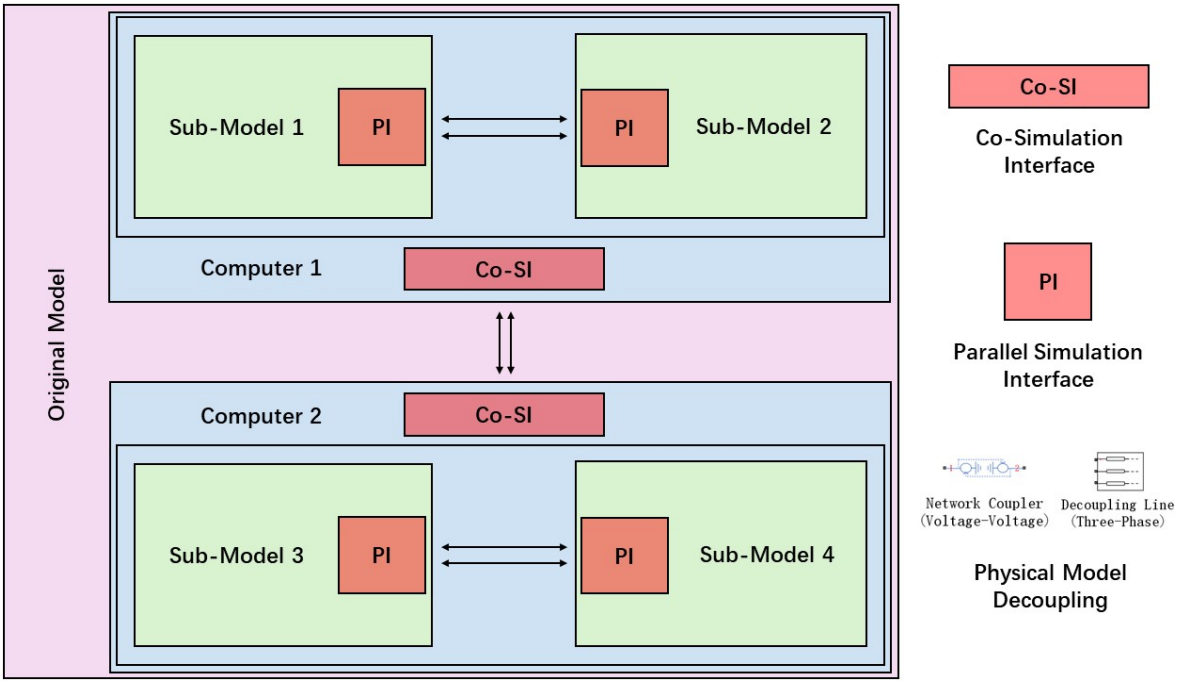

Figure 1.1: Project Framework.

1.1.1 Decoupling Method and Interface Design

Relaxation algorithms and transmission line modeling (TLM) are commonly used for circuit solving and analysis. The relaxation algorithm is based on Kirchhoff's laws, and its basic idea is to decompose the system into several subsystems, solve these subsystems independently using guessed initial conditions, and apply iterative methods to solve the equations [6]. The relaxation algorithm allows each subsystem to be solved independently in each iteration using approximate boundary values from adjacent subsystems, continuously updating these boundary values until a converged solution is obtained. Gauss-Seidel and Gauss-Jacobi are two typical relaxation algorithms. The key to implementing this algorithm is the partitioning of subsystems, ensuring that there are not too many tightly coupled components within a single subsystem [7].

Transmission Line Modeling (TLM), also known as bi-lateral delay line modeling [8], is an algorithm widely used in transmission line power systems. Its core idea is to achieve coupling between simulation units through physical time delays. Specifically, TLM divides the system into multiple simulation units, connects them via transmission lines, and introduces time delays. This method addresses the issue of time delays in variable exchange in non-iterative fixed-step parallel scheduling algorithms [9]. TLM only needs to compute the state at the current time point within each time step, while the information transmitted through the transmission lines includes a physically reasonable time delay, avoiding the need for frequent iterations to synchronize the states of different units [10]. As a result, the coupling between simulation units is naturally resolved within a single time step without requiring multiple calculations to adjust their states. Additionally, by introducing physical time delays, TLM effectively prevents instability issues caused by interactions between different simulation units, thereby enhancing the overall stability of the simulation.

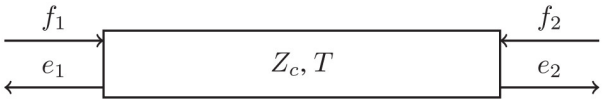

Figure 1.2: One-dimensional TLM element with characteristic impedance Z_c and time delay T.[11]

In this work, the interface design draws on the key concepts of the transmission line approach by introducing transmission line delays to solve the loop deadlock problem. According to this method, the decoupling points should be chosen at passive components in the system where voltage or current changes are slow. Therefore, we perform decoupling at the transmission line module. By doing so, the decoupled subsystems can be considered as being connected through constant voltage or current sources, which can significantly reduce errors caused by time delays [12], [13]. Transmission line delays introduce a fixed time delay in the data transmission path, causing the sending and receiving of data to be staggered in time, thereby breaking the synchronization dependency within the loop. This approach functions similarly to

1

adding a memory block module in the signal path, but by using time delays, we can avoid explicitly using memory blocks, thus making the simulation model more closely resemble the actual behavior of physical systems.

The Transmission Line Modeling (TLM) equations are derived from the plane wave solution to the wave equation:

$$F(s,q) = C_1 e^{\frac{sq}{a}} + C_2 e^{-\frac{sq}{a}} \tag{1.1}$$

Waveform variables are exchanged between interconnected simulation models with a specified time delay. In this work, the port delay references the propagation speed of uniform plane waves on the transmission line provided by the MATLAB transmission line (equivalent baseband) module [14]. For each sent and received data packet, the current time is used as a timestamp (obtained from the simulation timestep). Figure 1.3 illustrates the algorithmic process of implementing the introduced transmission line delay. In each simulation step, the current time is recorded as a timestamp when sending a data packet. Upon receiving a data packet, the receiver calculates the difference between the current time and the packet's timestamp. If the difference is less than the predetermined delay time (1.5 units of time in the figure), the receiver waits until the delay time has elapsed before processing the data. The arrows indicate the delay between the send and receive times. With this algorithm, deadlocks can be prevented in the Simulink model without relying on Memory blocks.

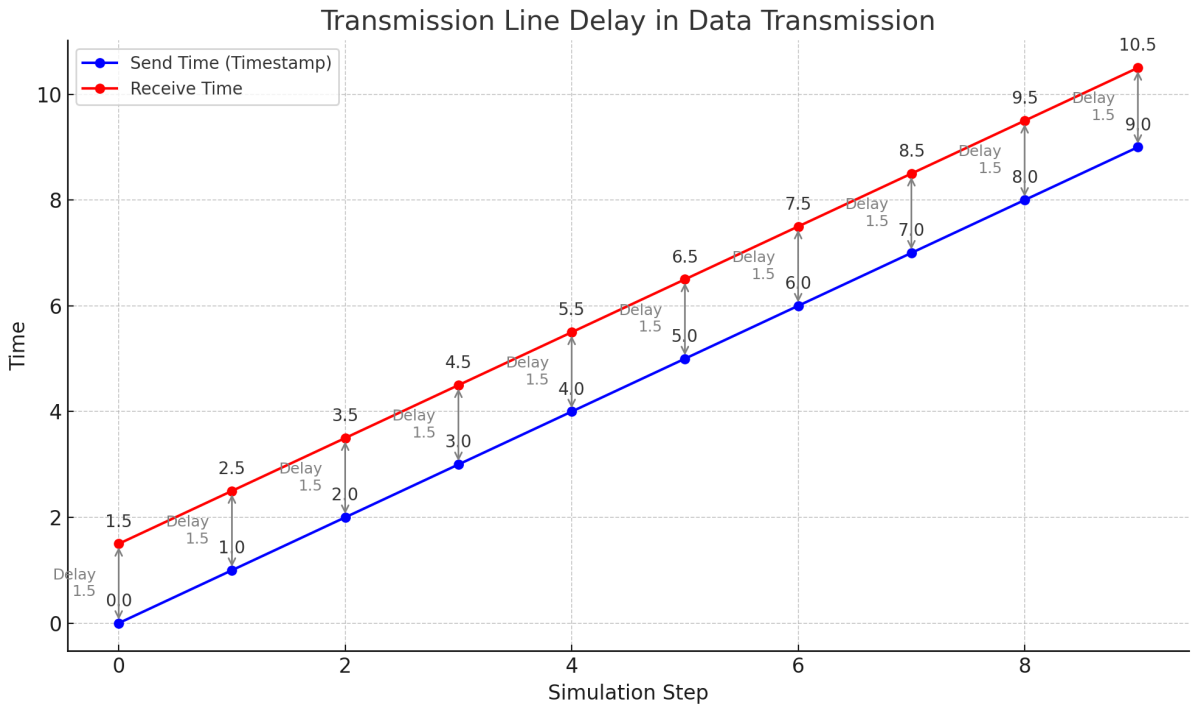

Figure 1.3: Transmission Line Delay Algorithm.

1.1.2 Model Decoupling

Most power transmission models in Simulink are based on Simscape physical systems. However, physical signal lines cannot achieve cross-model interaction. Therefore, it is necessary to decouple the physical system and convert the physical signals into Simulink signals for transmission across interfaces. As shown in Figure 1.4, the Decoupling Line (Three-Phase) module interrupts the physical transmission line while maintaining the ability to transmit signals. Essentially, the decoupling module is implemented internally based on Simulink® Goto/From modules, which only allow signal variables to be transmitted within the same model. However, we can create interfaces to store signals as global variables or use TCP/IP protocol to transmit signals, thereby enabling parallel simulation and co-simulation.

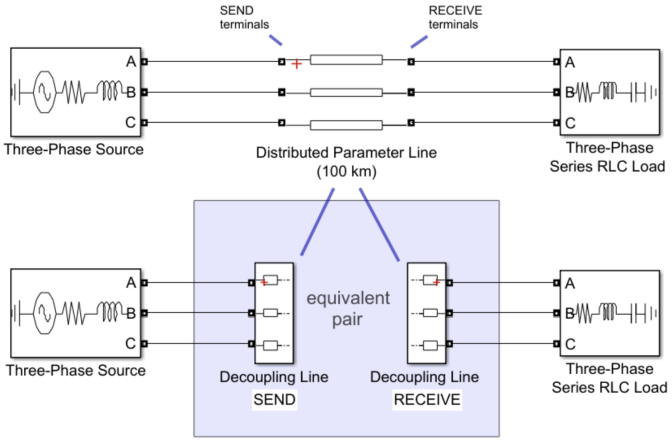

Figure 1.4: Simulink Model Decoupling. [15]

1.1.3 Communications Protocol

Matlab's Instrument Control Toolbox supports two host-to-host layer remote communication transport protocols: Transmission Control Protocol/Internet Protocol (TCP/IP) and User Datagram Protocol (UDP). TCP is a connection-oriented protocol responsible for breaking down byte streams into segments and reassembling them at the other end. In TCP transmission, the source and destination port fields together identify the two local endpoints of a specific connection. The ports, combined with their host's IP address, form a unique endpoint. Ports are used to communicate with the upper layers and to distinguish different application sessions on the host. Therefore, TCP is referred to as a stream-oriented protocol. TCP offers higher reliability, not only sorting data but also retransmitting lost data. TCP establishes a connection using a three-way handshake and disconnects using a four-way handshake. It uses sequence numbers and acknowledgment numbers to ensure the orderly and complete transmission of data and employs a window mechanism for flow control and congestion control.

1

On the other hand, UDP is a connectionless protocol that leaves reliability to be handled by the application layer. A UDP application prepares a data packet and sends it to the recipient's address without first checking if the recipient is functioning properly, and the recipient does not need to reassemble the data into a stream. Therefore, UDP is a packet-oriented protocol. The advantage of UDP is that it has much higher communication efficiency than TCP, significantly reducing end-to-end latency and enabling real-time data transmission. UDP does not require a connection to be established and directly sends data packets to the target address. Each data packet is independent, containing source port, destination port, length, and checksum information. Because UDP does not provide retransmission, sorting, or flow control, its implementation and operation are simpler, but the application layer must handle potential packet loss and disorder issues.

Feature	**TCP**	**UDP**
Reliability	High, ensures reliable data transfer through retransmission mechanisms	Low, does not provide retransmission mechanisms
Connection Type	Connection-oriented, requires connection establishment and termination	Connectionless, directly sends packets
Transfer Method	Stream-oriented, segments data into multiple packets	Packet-oriented, each packet is transmitted independently
Flow Control	Provides, controls data flow through a window mechanism	Does not provide
Congestion Control	Provides, uses algorithms like slow start and congestion avoidance	Does not provide
Transmission Efficiency	Lower, due to overhead from connection establishment, retransmission, and ordering	Higher, fast transmission with low latency

Table 1.1: Comparison of TCP and UDP Advantages and Disadvantages

In large-scale power transmission system co-simulation, UDP is the most commonly used transmission protocol. This is due to the characteristics of power systems, where simulations can tolerate a small amount of data loss without affecting the results, but require extremely low latency to avoid electromagnetic transients caused by the integration of different systems. Additionally, state estimation and fault detection require real-time data updates, making UDP's fast transmission feature the preferred choice.

However, in this project, TCP/IP remains the preferred signal transmission protocol. This is because the interface designed for this project has not completely separated the models into independent entities but maintains their connectivity on two computers through data interfaces. Inaccurate or missing data can easily lead to changes in signal precision, further reducing simulation accuracy. Moreover, TCP allows the use of timestamps to mark each piece of data, which facilitates the introduction of transmission delays and helps with further data calibration. By combining the handshake protocol with timestamps, we can maximize data integrity and ideally achieve simulation results identical to those of the original model simulation (e.g., intra-LAN communication).

Consideration needs to be given to how to address the high latency that can occur in TCP commu-

nication across a network. TCP estimates the round-trip time (RTT) by measuring the interval between sending a data packet and receiving an acknowledgment (ACK) [16]. However, due to the uncertainty of network delays, this estimation may not be accurate enough. This is why the timestamp option is added, allowing both sender and receiver to more precisely understand the transmission time of data packets. This helps to more accurately adjust the retransmission timeout (RTO)[17], thereby improving the reliability of data transmission. Additionally, in long-duration connections, the 32-bit sequence number of TCP may experience wraparound. This means that once the sequence number reaches its maximum value, it starts over, which can lead to confusion in packet delivery, as delayed old packets may be mistaken for new ones. Timestamps can help detect this sequence number wraparound, preventing erroneous packet reception. This enhances TCP's stability and reliability in high-bandwidth and long-latency networks.

Figure 1.5 illustrates the signal transmission reliability mechanism established by the project, where the delay specifically refers to the delay caused by network communication, rather than the transmission delay introduced by the TLM algorithm. In the co-simulation, we added an additional signal receiving and sending port to the original 3 ports (3-phase voltage/current signals). By using a clock module to receive the model's time data at each step as a timestamp for the data packets, and matching each timestamp at the receiving end, the delay is effectively eliminated.

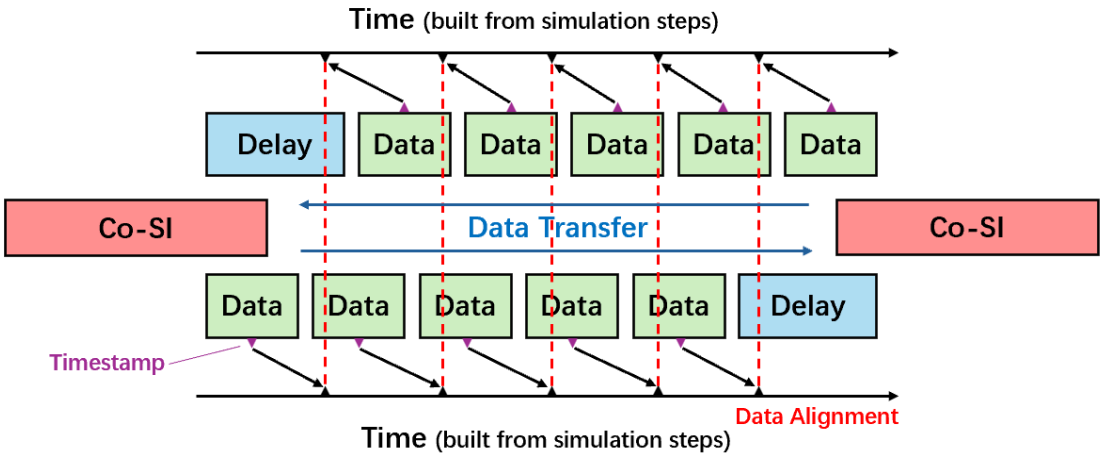

Figure 1.5: Timestamps for data alignment.

1

1.2 Parallel Simulation

1.2.1 Factors affecting simulation speed

In MATLAB/Simulink, simulation speed is influenced by several factors, including simulation duration, simulation step size (number of iterations), model scale and complexity, simulation mode, model compilation, and computer hardware. In this project, the co-simulation relies on interfaces to transmit electrical data, which inherently requires the discrete simulation to use a fixed step size. This leads to a significant issue: the step size itself affects the simulation speed, and coupled with the transmission time and signal delay of the interface, achieving high-precision simulation in large-scale complex models becomes an extremely time-consuming task. This inefficiency in hardware resource usage contradicts the original intention of co-simulation and reduces its value in engineering applications.

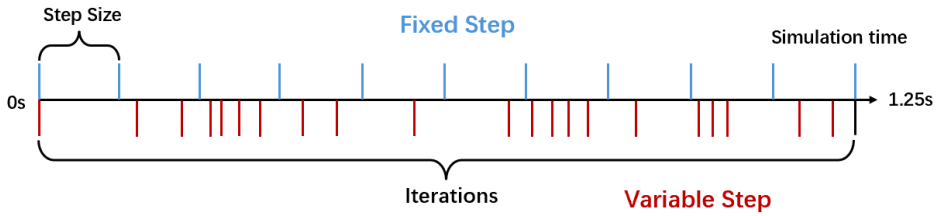

Figure 1.6: Difference between fixed and variable step size.

To address this issue, we need to improve the simulation speed without changing the simulation step size. Since simulation accuracy is directly proportional to the step size, a trade-off between precision and efficiency is necessary. One optimization approach is to reduce the computation time per time step through parallel simulation, thereby accelerating the overall simulation speed. Parallel simulation can fully utilize the computational power of multi-core processors, breaking down large and complex models into multiple sub-models for parallel computation, significantly enhancing simulation efficiency.

1.2.2 Implementing Parallel Simulation in Matlab

MATLAB's Parallel Computing Toolbox provides support for parallel simulation of models. For Simulink models, the parsim or batchsim functions allow the use of object arrays to create simulation sets using Simulink.SimulationInput to run multiple simulations [18]. These two functions essentially run models on multiple MATLAB® workers within a parallel pool. However, the difference is that parsim is primarily used for parallel simulations on local multicore machines or small-scale clusters, making it suitable for quickly deploying parallel tasks on a single computer. As shown in the figure 1.7, in the parsim workflow, the client directly communicates with multiple workers, assigning simulation tasks directly to them.

In contrast, batchsim is designed specifically for distributed computing cluster environments, capable of allocating and managing simulation tasks across large-scale computing clusters. In the batchsim workflow, the client first communicates with a head worker, which coordinates and distributes tasks to other workers, making it suitable for large-scale simulations that require extensive computing resources [19]. Therefore, in this work, the parsim function was used to construct parallel simulation execution files to fully utilize local computing resources and improve simulation efficiency.

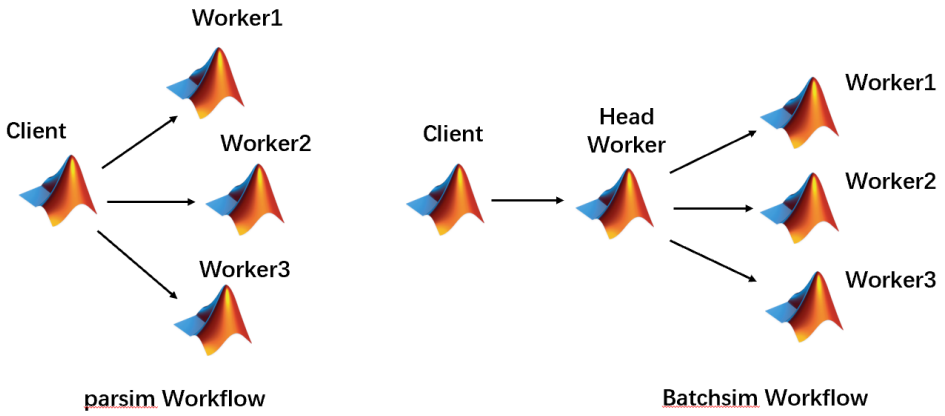

Figure 1.7: Difference between parsim and batchsim workflow.

1.2.3 Parallel Simulation Planning

Figure 1.8 shows the basic framework of parallel simulation, where a large-scale simulation task is divided into multiple independent subtasks, each of which can be executed in parallel on different cores of a computer [20]. This method reduces the complexity of individual tasks and fully utilizes the computational power of multi-core architectures, thereby accelerating the overall simulation process. In this example, we divide the 116-Bus, 28-Power Plant Network power transmission model into four 29-Bus submodels [21]. For the power transmission model, the decoupling tool in Powergui can be used to decouple selected transmission lines into two independent interfaces, allowing electrical quantities (three-phase voltage/three-phase current) to be transmitted through these interfaces. By setting appropriate global variables, each submodel can operate independently. A key point is that the license provided by MATLAB's Parallel Computing Toolbox allows the parsim function to initiate parallel simulations across the computing cluster after the submodel initialization is complete. Otherwise, the simulation will run in series, making subsequent co-simulation impossible.

To verify whether parallel simulation improves simulation speed, a simple method is to use MATLAB's Performance Advisor to observe the model's initialization time and simulation duration. However, in this project, the partitioned submodels need to be co-simulated on two computers. Due to the handshaking mechanism, the total time of co-simulation is constrained by the transmission speed of interface

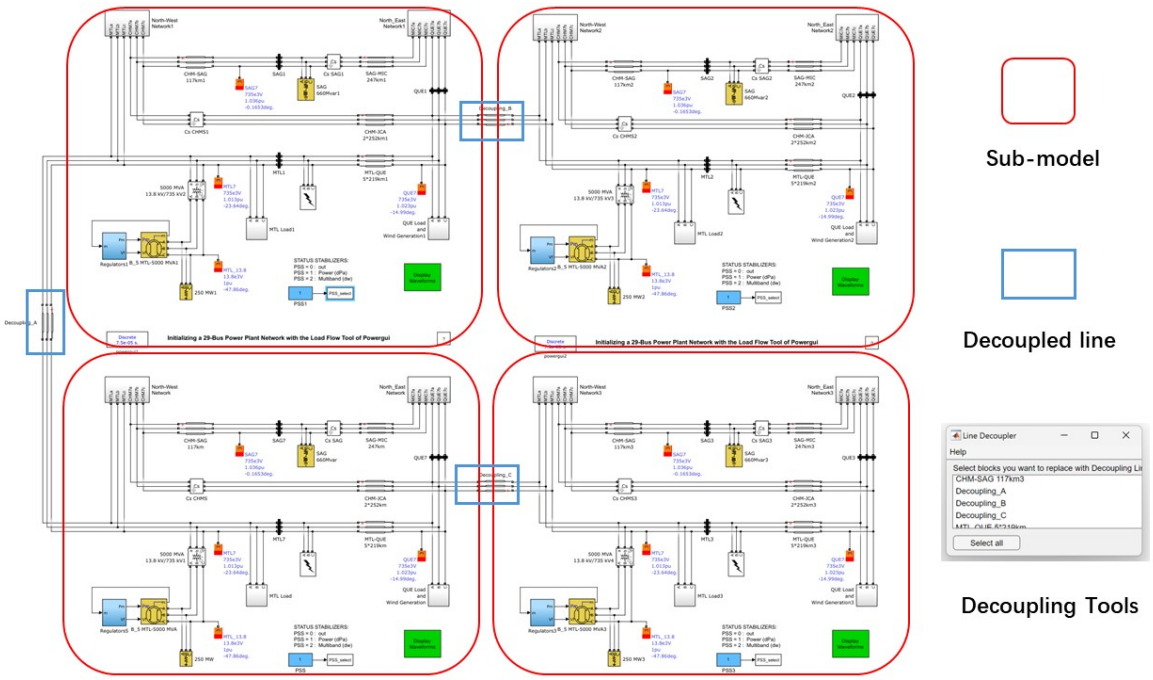

Figure 1.8: Parallel simulation framework.

information. Therefore, using only the change in simulation time to demonstrate the advantages of parallel simulation is not sufficiently comprehensive. To address this, we developed a CPU core utilization monitoring program in MATLAB to validate the benefits of parallel simulation by comparing changes in CPU core utilization. This approach effectively demonstrates how parallel simulation can utilize multicore processor resources to enhance overall simulation efficiency.

As shown in Figure 1.9, a graphical user interface (GUI) was implemented using a MATLAB program to monitor the CPU usage of the computer, and when run it will display the CPU usage of each core in real time as well as the average usage over a 10 second period. Since MATLAB can interact with .NET libraries, this makes it possible to use the system functions provided by Windows to obtain low-level hardware information. NET's System.Diagnostics.PerformanceCounter class is used in this program to access the performance counter. By creating a PerformanceCounter object for each CPU core, performance counters for the monitored CPU cores can be generated. The processors of the two computers used for the co-simulation in this project are AMD Ryzen 9 8945HS and Intel Core i7-13900, both with 16 cores. The left image in Figure 1.9 shows the processor monitoring screen of the computer's task manager, and the right image shows the Matlab GUI interface.

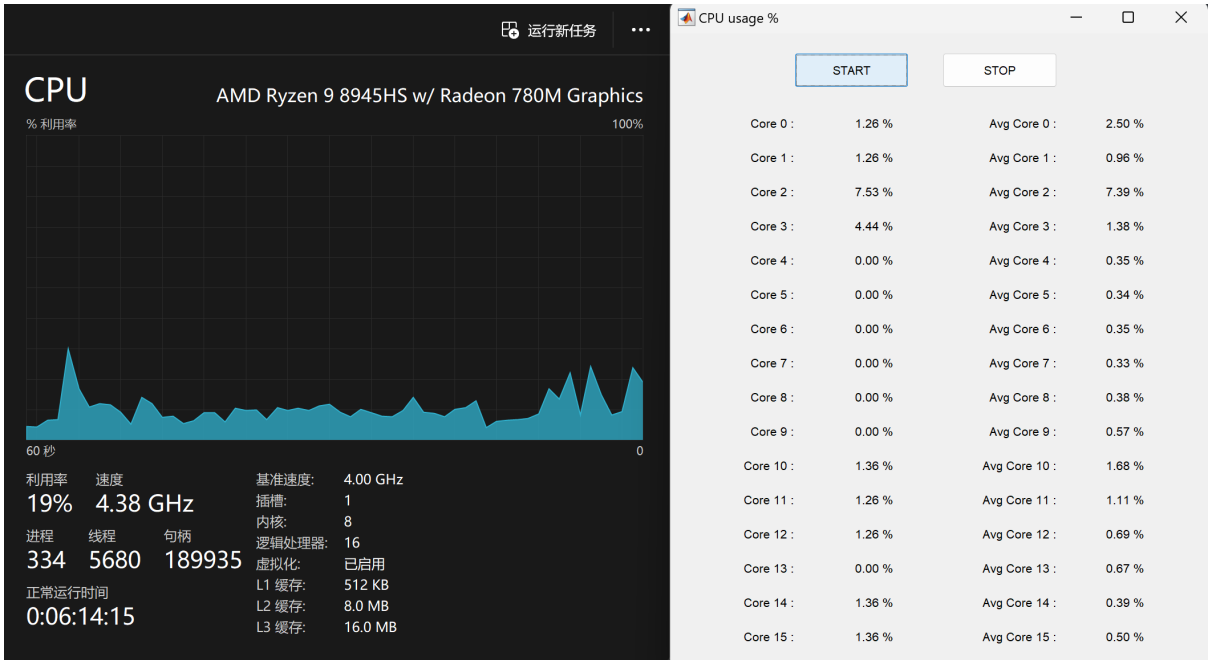

Figure 1.9: CPU utilization monitoring.

1.3 Scope of the Thesis

Chapter 1 introduces the basic framework of the co-simulation implemented in this project, explaining the communication mechanisms, interface algorithms, and the principles and feasibility of co-simulation and parallel simulation.

Chapter 2 describes how to separate the grid-connected inverter model into sub-models and realize co-simulation through the interface. The accuracy of the co-simulation is verified by comparing the simulated waveforms at both ends of the interface and the output characteristics of the inverter.

Chapter 3 describes how to realize co-simulation in large-scale multi-bus power systems. The chapter also describes how to combine co-simulation with parallel simulation in order to rationalize the division of multi-bus systems and to reduce the simulation time. The outputs of the simulations are compared and the feasibility of using parallelism to accelerate large-scale co-simulation systems is verified.

Chapter 4 introduces how to build a visualization application in MATLAB as a co-simulation platform to enable quick command execution and efficient comparison of simulation results.

Finally, Chapter 5 provides a comprehensive conclusion of the project's achievements and outlines plans for potential future work.

1

2

Co-simulation of grid-connected inverter systems

Contents

2.1 Grid-Connected Inverter Model and Partitioning Strategy 13
2.1.1 Original Model Introduction 13
2.2 Simulation Process 15
2.2.1 Model Setup and Interface Compilation Ideas 15
2.2.2 Network Connection 17
2.2.3 Simulation Workstation 17
2.3 Simulation Results 19
2.3.1 Co-simulation Results in LAN 19
2.3.2 Co-simulation Results Across Campus Network 22

2.1 Grid-Connected Inverter Model and Partitioning Strategy

2.1.1 Original Model Introduction

The test case uses the optimal current control model of a three-phase grid-tied inverter from MATLAB's example models, as shown in the figure 2.1. The control algorithm of this inverter model employs an observer-based linear quadratic regulator (LQR) strategy and replaces integral action with an observer to ensure zero steady-state error. Since the model is controlled based on feedback signals, we can split it into two independent sub-models. The red box in the figure includes the control part on the inverter side, while the blue box covers the rest of the circuit, achieving precise control by returning

feedback signals (green box) to the controller. This method allows us to separate the controller from the circuit and exchange signals through interfaces at each simulation step to achieve co-simulation. This approach effectively isolates the inverter-side control algorithm provided by vendors from the power system operations without disclosing model details to either party.

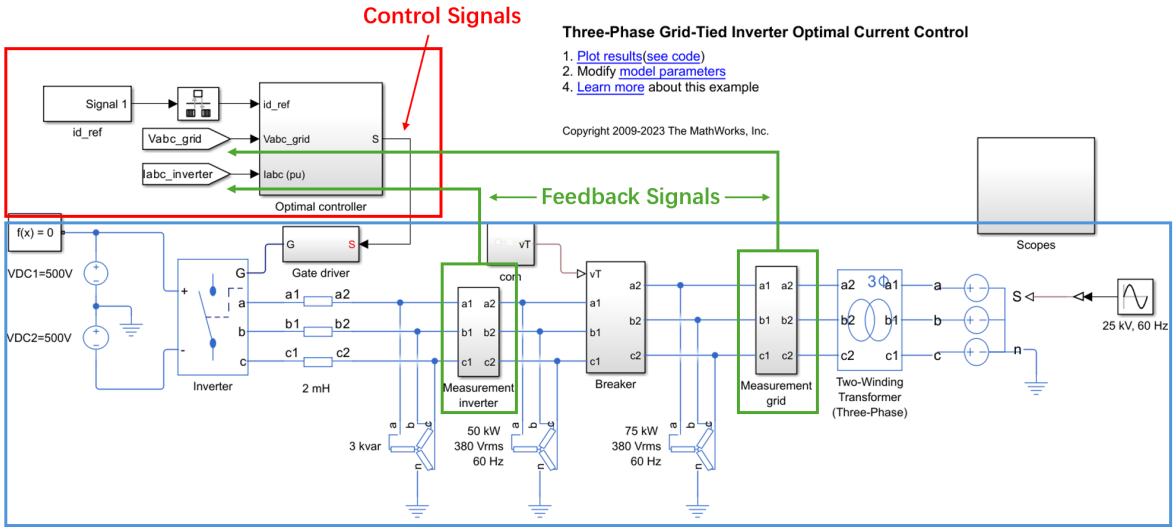

Figure 2.1: Original Grid-Connected Inverter model.

After running the simulation, the inverter connects to the grid at 0.15 seconds. Subsequently, at 0.2 seconds, the inverter increases the active power supplied to the grid, as shown in the figure 2.2. This provides a good reference for subsequent co-simulation, as comparing the transient responses of the inverter before and after separation can assess the feasibility and accuracy of the co-simulation.

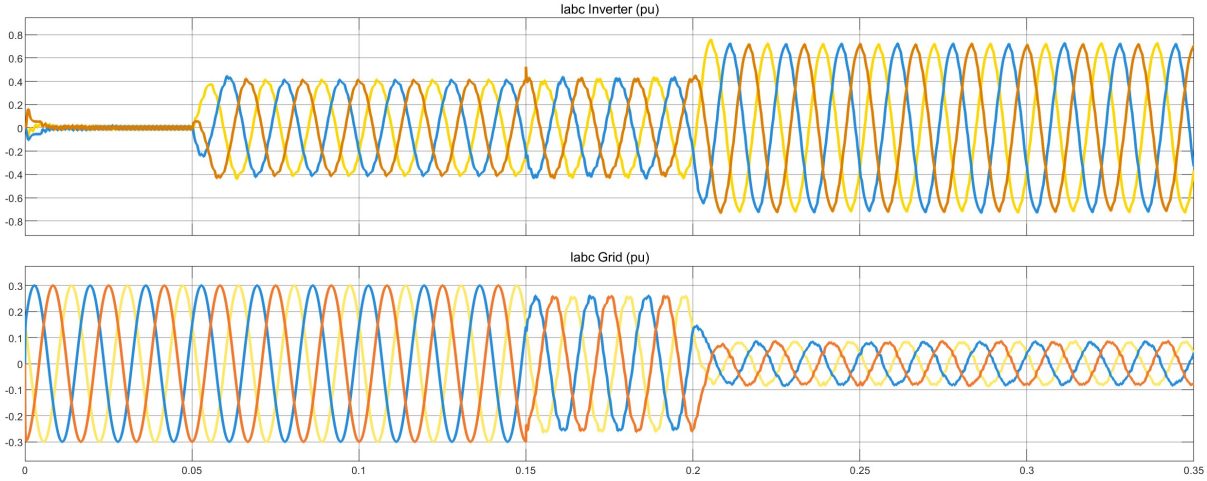

Figure 2.2: Enter Caption

2.2 Simulation Process

2.2.1 Model Setup and Interface Compilation Ideas

After splitting the original model, we obtained two sub-models, shown in Figure 2.3. We use MATLAB's S-Function module to create communication interfaces, naming them Server and Client in the two sub-models. MATLAB S-Function is used to implement custom functions and extend models in Simulink, allowing users to write custom code in languages such as MATLAB, C, C++, or Fortran to achieve functions that the native Simulink modules cannot. Combined with the common communication protocols supported by the MATLAB Instrument Control Toolbox (such as VISA, GPIB, TCP/IP, and UDP), we can implement communication between interfaces. Both interfaces have the capability to send and receive signals, with the Server responsible for receiving the grid's feedback three-phase current and voltage signals and transmitting them to the Client, as well as receiving control signals returned by the Client. The Client does the opposite. To keep the model as neat and tidy as possible, we use MATLAB's Goto/From modules to pass signals within the sub-models. In practice, $V_{\text{abc\_grid}}$ and $I_{\text{abc\_inverter}}$ are collected by `Measurement_inverter` and `Measurement_grid`, with the corresponding subsystems hidden in the figure. In this project, both models are run in MATLAB version 2024a. The original model used for testing can be opened in the command window with the command `openExample('simscapeelectrical/GridTiedInverterOptimalIExample')`. It is important to note that this example model was added in version 2023b, so it can only be accessed using MATLAB version 2023b or later. Earlier versions do not support this shortcut command.

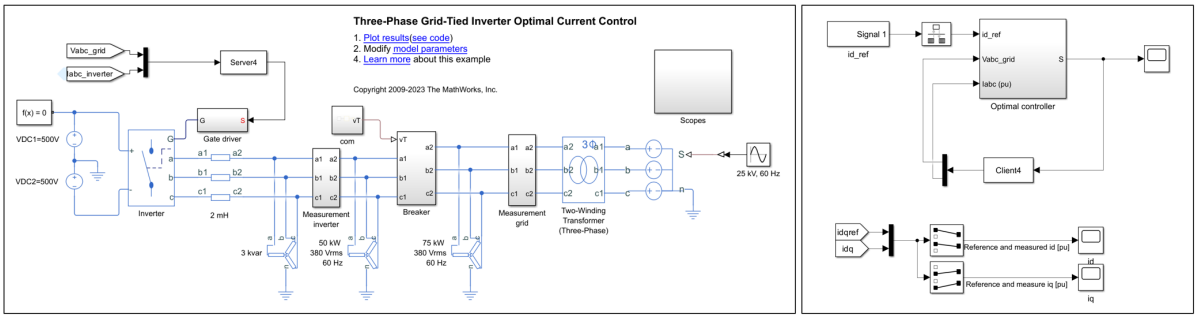

Figure 2.3: Submodels communication interfaces.

The interface code used in the S-Function is shown in Appendix A. The basic logic of this module is as follows: First, it initializes and configures two TCP servers for receiving and sending data. Each data packet contains six double-precision floating-point numbers used to transmit three-phase voltage and current signals or their three-phase control signals. (In subsequent simulations, the data packet size

will increase to seven to include a timestamp for labeling each data packet.) During the simulation, the server starts first and listens for control signals from the client. When a valid control signal is received, it is updated and passed to the Simulink model output. If an error occurs in communication, the module will attempt to reconnect to ensure data transmission stability, unless the set timeout is exceeded. Additionally, a persistent variable `last_valid_control_signals` is used to store the last valid control signals, ensuring that the system continues to output valid control signals in case of communication failure, maintaining stable operation. Attention should be paid to the fact that the sampling time set in the communication interface must be the same as, or a multiple of, the fixed step size of the simulation model.

The `tcpip` object and `fopen` function used in the code implicitly handle the TCP three-way handshake process, so there is no need to implement it at the application level [22]. When the `fopen` function is called, the underlying TCP protocol automatically manages the three-way handshake process, as illustrated in Figure 2.4. During the first handshake, the client sends a SYN (Synchronize Sequence Number) packet to the server, indicating a request to establish a connection. This packet includes the client's initial sequence number (`seq = x`). In the second handshake, the server receives the SYN packet and must reply with a SYN-ACK packet. This packet acknowledges the client's sequence number (`ack = x + 1`) and includes the server's own initial sequence number (`seq = y`). In the third handshake, the client receives the SYN-ACK packet and sends an ACK (Acknowledgment) packet back to the server, confirming the server's sequence number (`ack = y + 1`). At this point, the connection is established, and both parties enter the ESTABLISHED state, ready to start data transmission [23]. The three-way handshake ensures that the connection between the client and server is reliable, bidirectional, and that both parties are ready for data transmission.

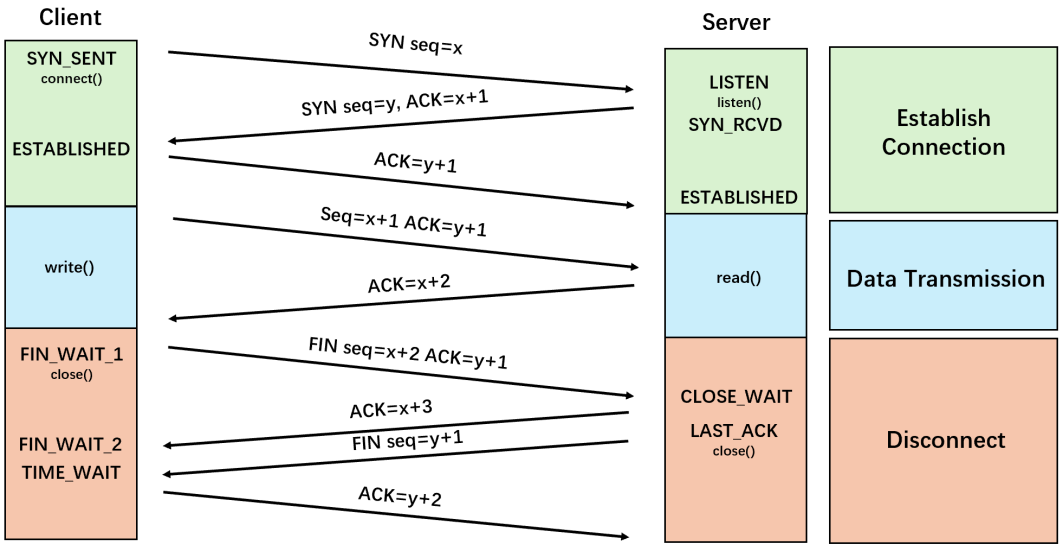

Figure 2.4: Three-way handshake process using the fopen function.

2.2.2 Network Connection

The collaborative simulation will be conducted under two scenarios: the first scenario is conducted within the local area network (LAN) of the apartment, and the second scenario involves establishing a connection between the apartment and the Imperial College campus. As shown in the figure 2.5, the apartment is located in Hammersmith, London, while the Imperial College campus is in South Kensington. The first scenario, based on LAN communication, has almost no delay, so theoretically, the results of the collaborative simulation should be consistent with those of the original model simulation. The second scenario involves cross-network connections, which may introduce some communication delays.

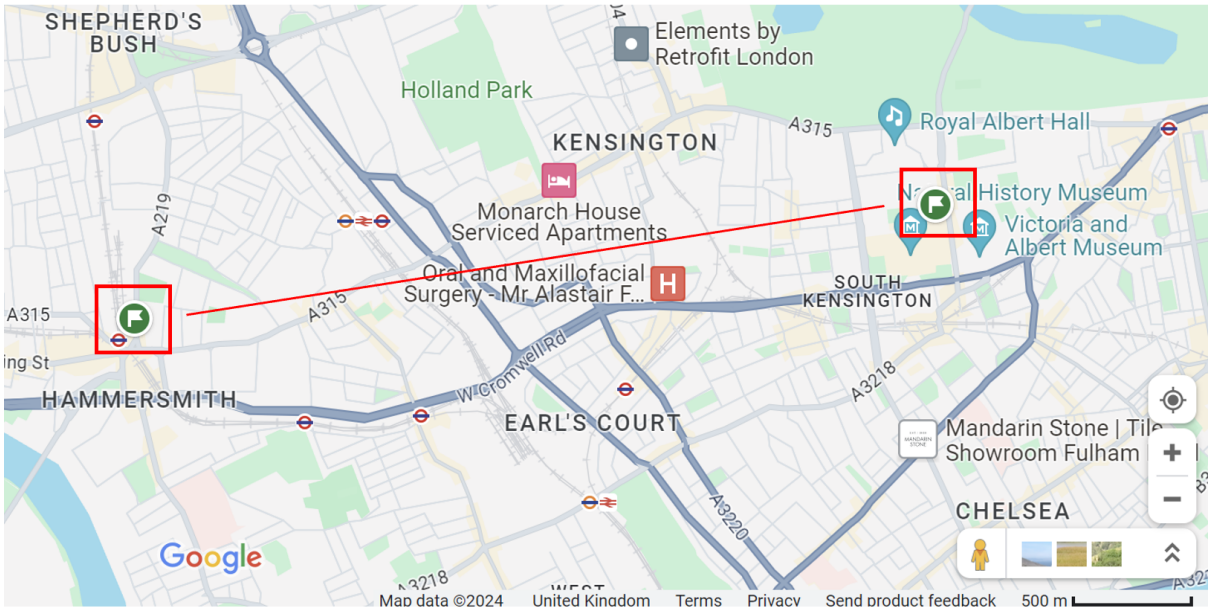

Figure 2.5: Network connection between apartment and the Imperial College Campus

2.2.3 Simulation Workstation

In this project, two personal computers are used as simulation workstations. The portability of the laptops provides convenience for setting up cross-network connections. As shown in Figure 2.6, the two sub-models will run independently on the two computers and communicate only through interfaces. When testing cross-network connections, one of the laptops will be taken to the Imperial College campus to establish the connection, while the other computer will be operated remotely. In the figure, the white laptop is equipped with an AMD Ryzen 9 8945HS CPU, and the black laptop is equipped with an Intel Core i7-13900. The simulation mode is set to 'normal', meaning that no compilation acceleration is applied. The reason for this setup is that in normal mode, we can use code instructions and the diagnostic viewer to monitor the data being transmitted and received at each step through the interfaces, allowing us to determine whether the co-simulation is functioning correctly.

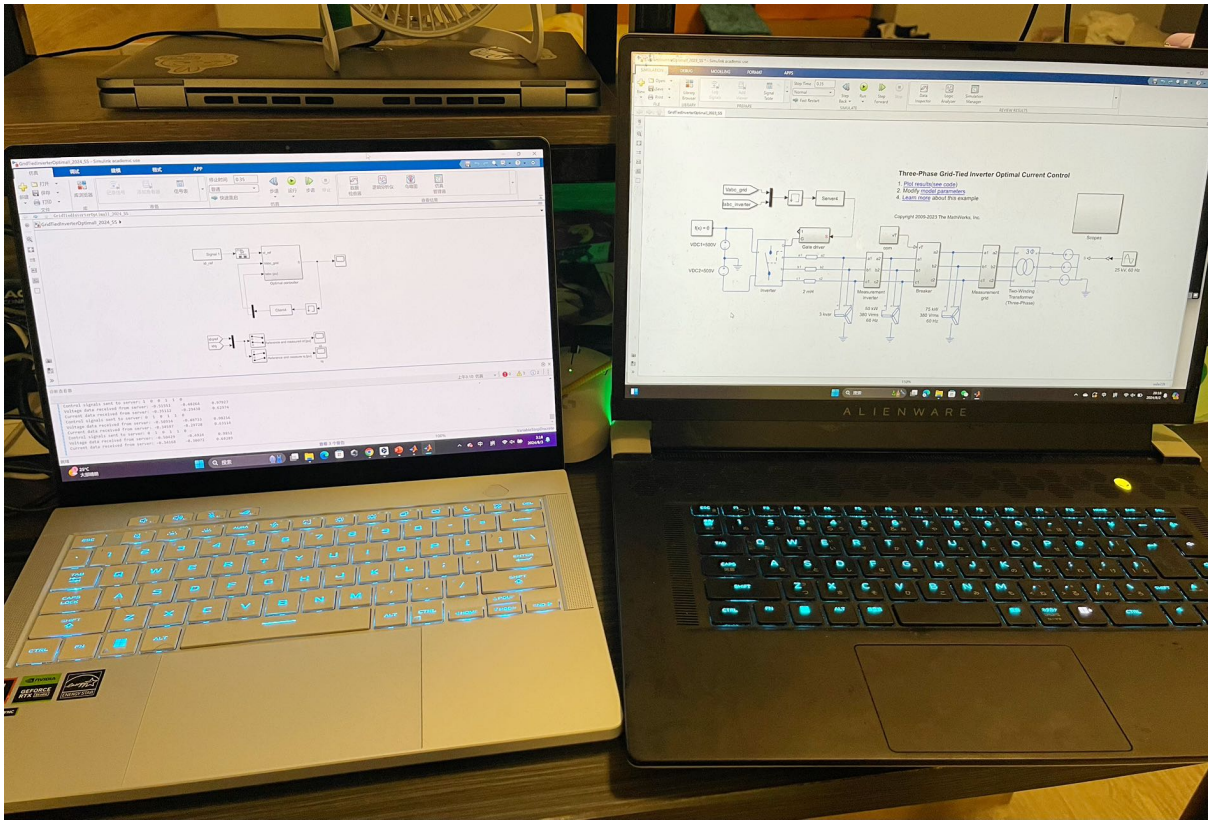

Figure 2.6: Simulation Workstation.

When running the simulation, the server-side submodel must be started first because the interface logic involves the server listening to the client. Both interfaces initially have a signal value of 0. After establishing the connection, they send the collected data in the next step, as shown in Diagnostic Viewer in Figure **??**. The server sends three-phase current and voltage signals with precision up to five decimal places. The client sends a square wave control signal, so the signal values are either 0 or 1. Since the submodels have the same step size and simulation time, the simulation will complete at the same time. After the simulation is complete, the results can be observed using the Scope set up in the submodel.

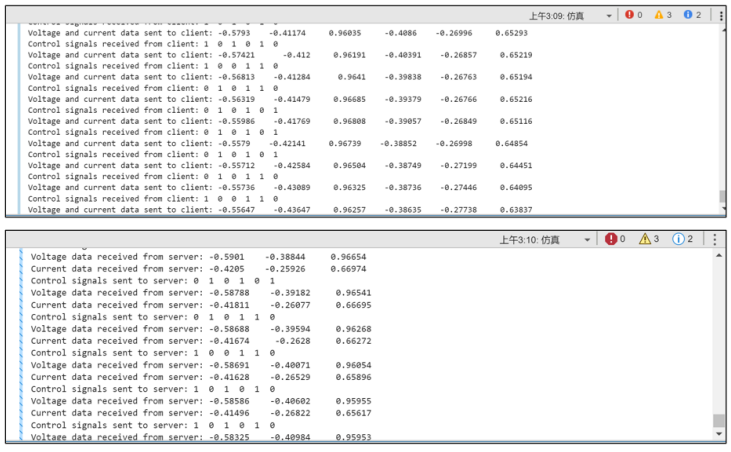

Figure 2.7: Diagnostic Viewer

2.3 Simulation Results

2.3.1 Co-simulation Results in LAN

Figures 2.8, 2.9, and 2.10 respectively show the comparison of inverter voltage, grid voltage, and synchronization results between the original model and the co-simulation model, the comparison of inverter current and grid current, and the comparison of inverter and grid power dynamics. The results indicate that the simulation results have almost no error compared to the original model. This is because the current tests are conducted in a local area network environment where communication latency is extremely low. When the step size and sampling time of the sub-models are consistent with those of the original model, the only factor contributing to errors is the precision of the model interface. In this simulation, the sampling time is 5e-06, and all models in the co-simulation use a fixed step size with the interface accuracy preserved to five decimal places, while the original model uses a discrete variable step size. The simulation results are consistent with the operating logic of the original model, i.e., at 0.15 seconds, the inverter is connected to the grid. Then, at 0.2 seconds, the inverter increases the active power supplied to the grid.

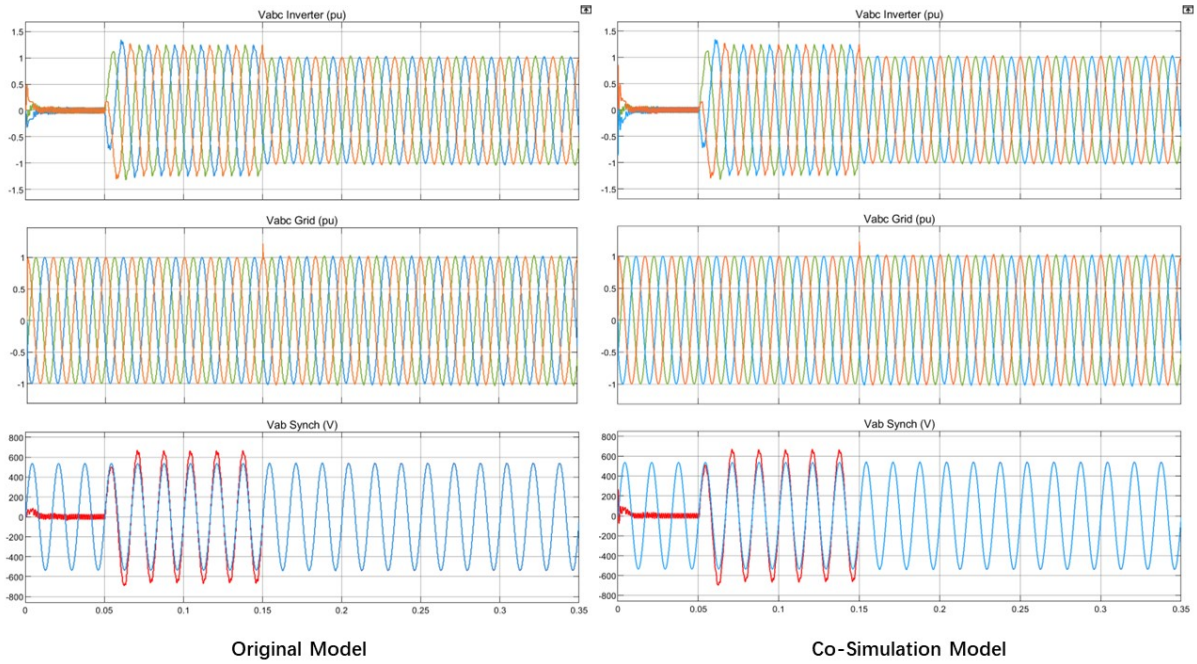

Figure 2.8: Comparison of inverter voltage, grid voltage and synchronization results between the original model and the co-simulation model.

To more accurately analyze the errors in the co-simulation, the Simulink built-in Simulation Data Inspector can be used to conduct a detailed comparison and evaluation of the generated waveforms. The Simulation Data Inspector allows for the comparison of results from different simulation models and

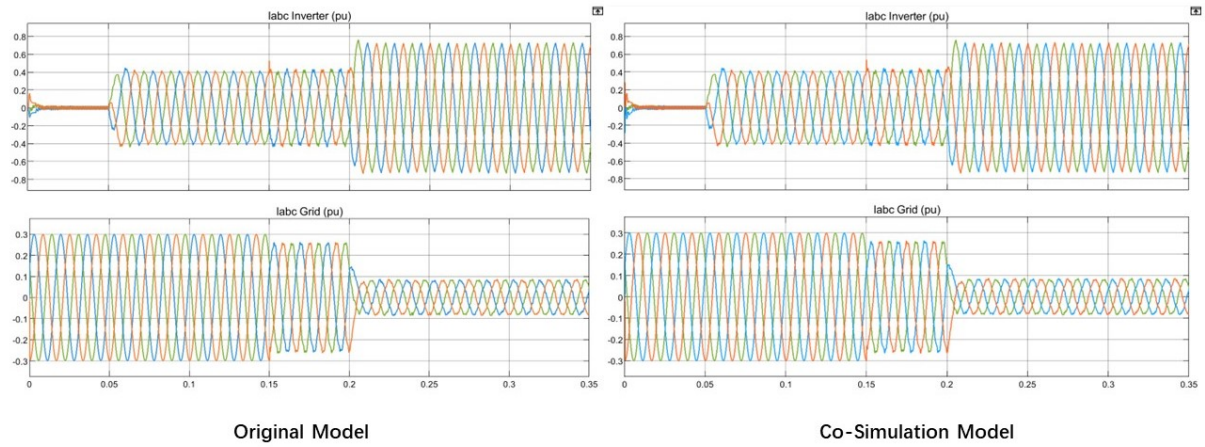

Figure 2.9: Comparison of inverter current and grid current between the original model and the co-simulation model.

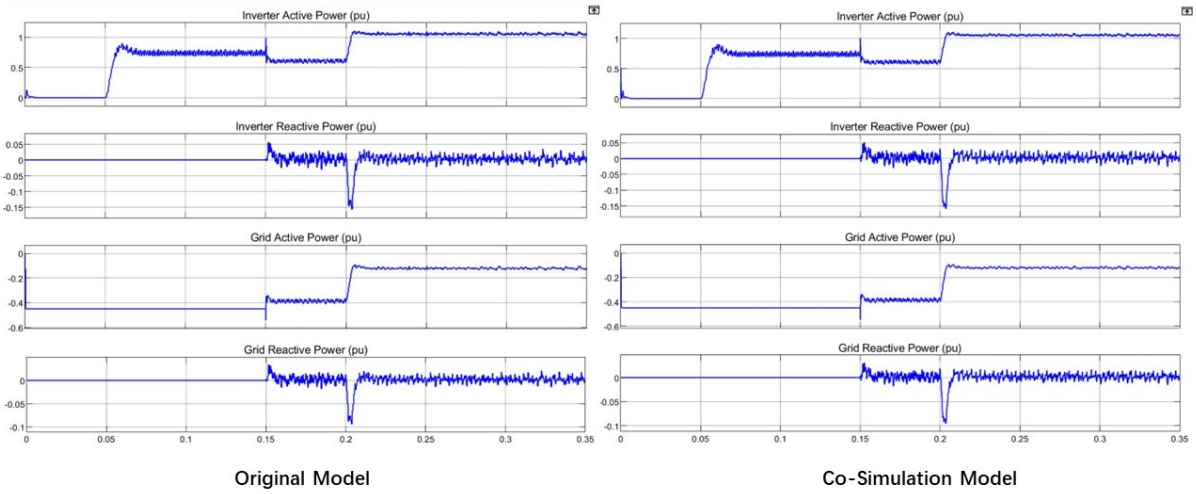

Figure 2.10: Comparison of Inverter and Grid Power Dynamics in Original and Co-Simulation Models

automatically calculates the maximum difference between them. Thus, we can run the original model on the server side and compare it with the co-simulation results. Due to the large number of comparison images, we will only analyze the Inverter active power dynamics and the grid current results, as shown in Figures 2.12 and 2.11. The remaining images can be found in AppendixB. Additionally, Table 2.1 was created to show the maximum error and total error percentage for each simulation result versus the original model simulation.

In Figure 2.11, it can be observed that there is no error between the simulation and the original model before 0.15 seconds. At 0.15 seconds, the inverter connects to the grid, which is a sudden event causing dynamic changes in the system, such as transient current surges and voltage fluctuations. At this point, the co-simulation begins to show errors. It is speculated that these errors may arise from data exchange delays and insufficient interface resolution to capture the rapidly changing signals during the system's dynamic changes, leading to an accumulation of simulation errors in subsequent stages.

Figure 2.12 shows the active power curve of the inverter, where it is evident that the active power error is greater than the current error and is present from the beginning of the simulation. This is because the calculation of active power involves multiple parameters (voltage, current, and phase angle), and it is therefore affected by the combined influence of errors in these parameters. Although each parameter's error may seem small, these errors can accumulate and amplify in the power calculation. Consequently, during dynamic events such as when the inverter connects to the grid, the system's transient response and instability are more prominently reflected in the changes in active power, leading to more noticeable errors in co-simulation. In this test, the accuracy of the active power simulation can serve as an important benchmark for evaluating the calibration precision of the co-simulation.

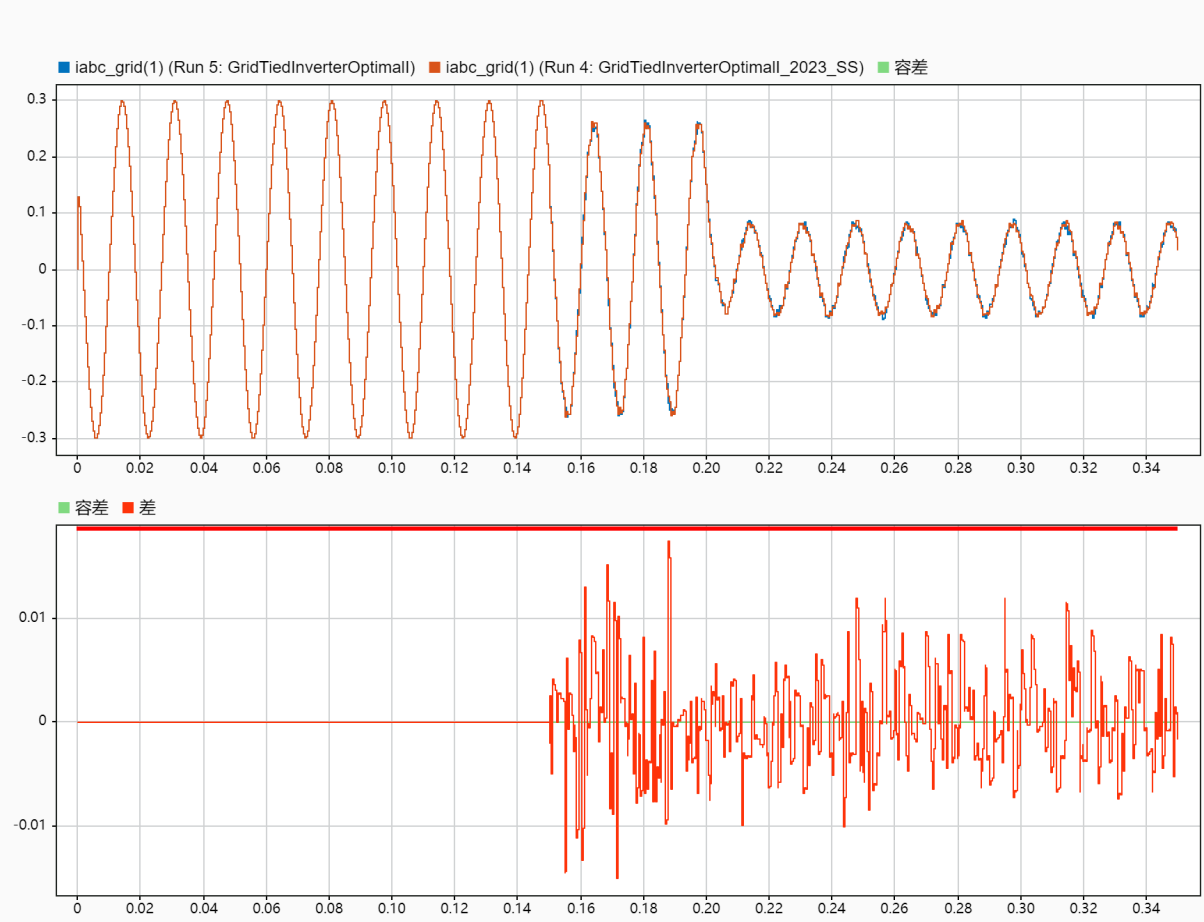

Figure 2.11: Comparison of original model and co-simulation model of grid three-phase current (One of the three phases).

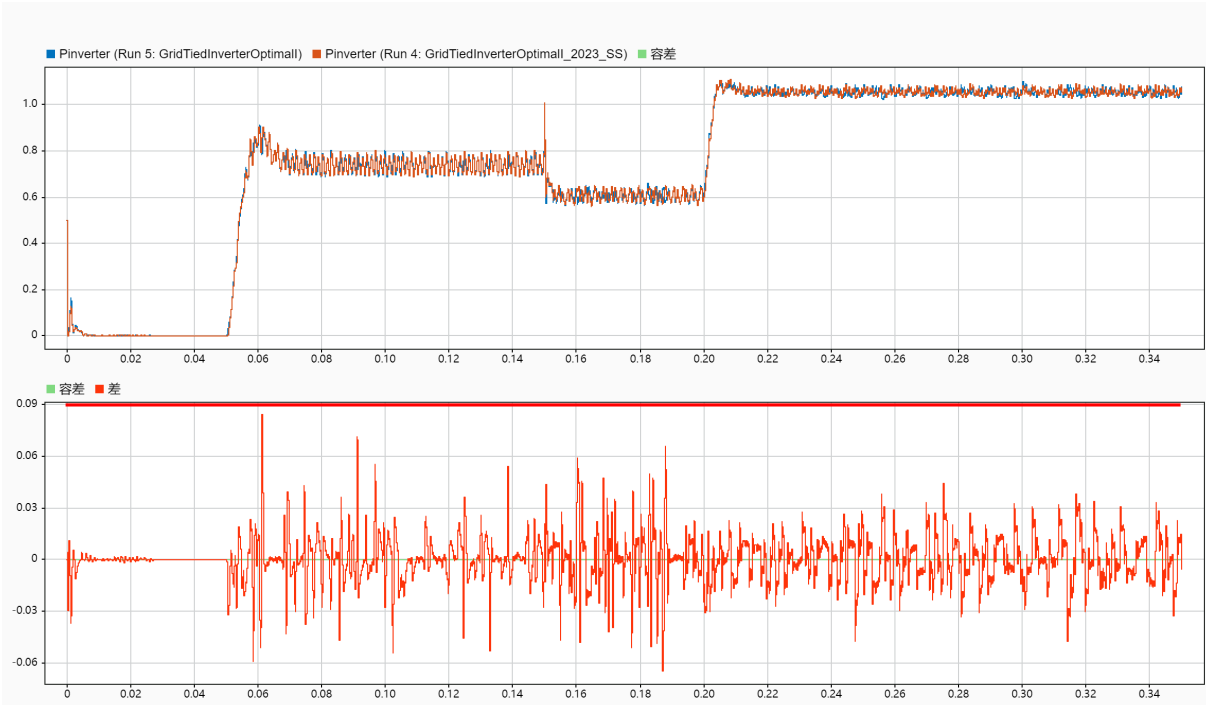

Figure 2.12: Comparison of original model and co-simulation model of inverter active power dynamic.

Signal Source	Maximum Error(p.u.)	Total Error in Percentage(%)
IabcInverter(a)	0.05	2.33
IabcInverter(b)	0.06	2.74
IabcInverter(c)	0.07	2.95
VabcInverter(a)	0.03	1.84
VabcInverter(b)	0.03	1.85
VabcInverter(c)	0.04	1.91
IabcGrid(a)	0.02	1.18
IabcGrid(b)	0.02	1.36
IabcGrid(c)	0.02	1.14
VabcGrid(a)	0.01	0.59
VabcGrid(b)	0.01	0.59
VabcGrid(c)	0.01	0.59
Pinverter	0.08	3.75
Qinverter	0.04	2.19
Pgrid	0.07	3.36
Qgrid	0.04	2.01

Table 2.1: Error analysis of each simulated signal compared to the original model results (LAN).

2.3.2 Co-simulation Results Across Campus Network

The basic setup of cross-network co-simulation models is the same as that of the LAN scheme. Although cross-network transmission introduces some delay, the timestamp method used (as discussed in the Introduction) effectively mitigates the impact of network latency. Figures 1, 2, and 3 show the differences in results between the co-simulation and the original model simulation. It can be seen that, compared to co-simulation within a LAN, cross-network co-simulation exhibits larger simulation errors, which are

likely due to the accumulation of errors caused by packet loss during data transmission. The specific times and frequencies of packet loss can be observed from the diagnostic viewer, as shown in Figure 2.13. According to the interface logic, when data is lost, the system uses the simulation data from the previous time step for output. The network delay during the simulation fluctuated between 19ms-41ms. In addition, by viewing the time array obtained from the logout data, a total of 106 packets were lost (out of a total of 70,001), which can be calculated to give a packet loss rate of 0.15% for this period of simulation.

```
Voltage and current data sent to client: 0.58108    -0.51964     0.99649    0.035728       2.641     -1.7933
Simulated packet loss during receiving: using last valid control signals.
Voltage and current data sent to client: 0.58108    -0.51964     0.99649    0.035728       2.641     -1.7933
Control signals received from client: 0  1  1  0  1  0
```

Figure 2.13: Package losses during simulation.

Figures 2.14, 2.15, and 2.16 show the electrical transients captured by Scopes for the grid and inverter during the simulation. It can be seen that, unlike the simulation within the LAN, the results of the cross-network co-simulation have some errors with the simulation results of the original model. These discrepancies often occur at the peaks and valleys of the three-phase circuit or voltage waveform, as these are regions of rapid transient changes. If the sampling rate is not sufficiently high, waveforms at these points are more prone to distortion. Additionally, transmission delays exacerbate the impact on the accuracy of these transient regions. As shown in Figure 2.15, after 0.2 seconds, the transient changes in the grid current are more pronounced than before 0.2 seconds, leading to more severe distortion. Moreover, packet loss can further amplify these errors. Although our interface logic compensates for lost data by using the previous step's data, packet loss still leads to incomplete data transmission, which accumulates and affects the accuracy of the simulation results. These errors in the three-phase currents and voltages accumulate further in the power curves of the inverter and the grid, leading to more serious distortions in the active and reactive power curves than in the voltage and current curves. In particular, it can be seen in Fig. 2.16 that significant fluctuations in reactive power occur after 0.2 seconds.

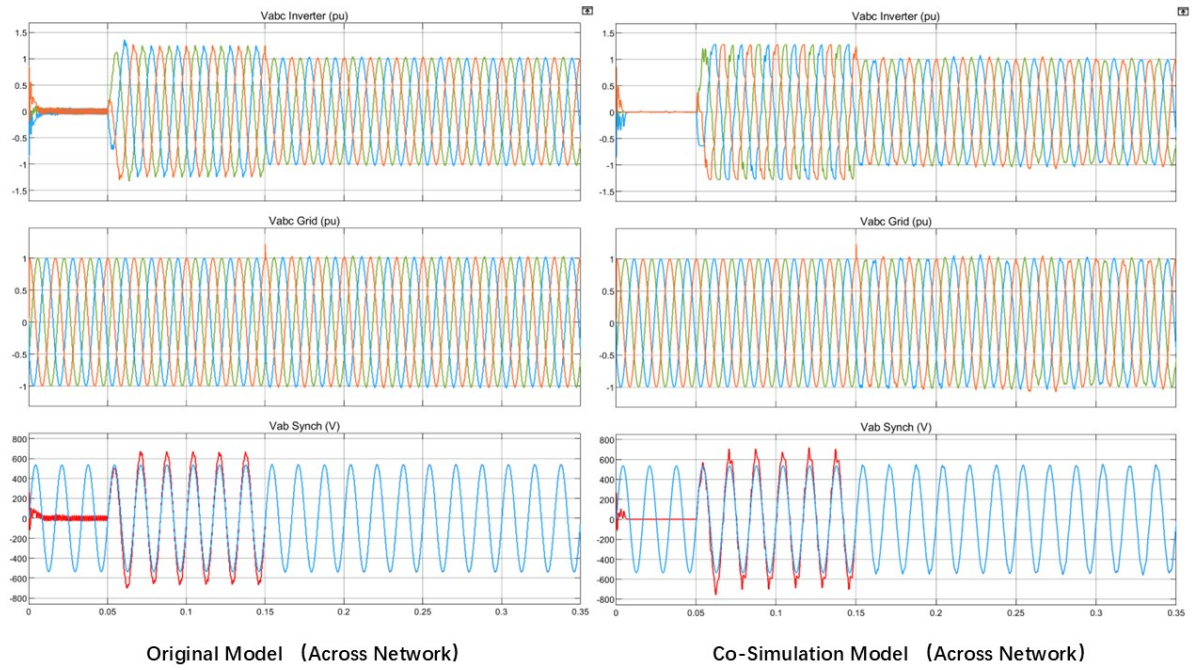

Figure 2.14: Comparison of inverter voltage and grid voltage between the original model and the co-simulation model (Across network, p.u.).

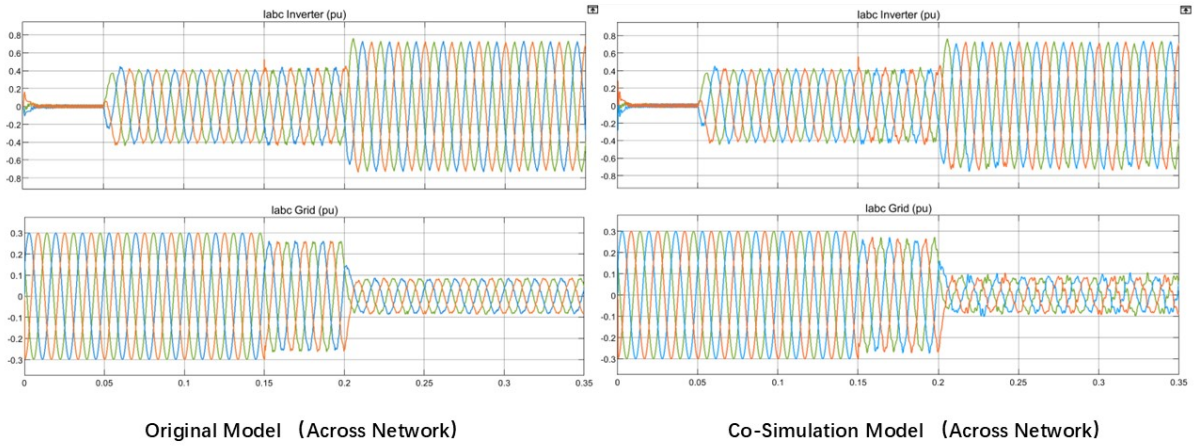

Figure 2.15: Comparison of inverter current and grid current between the original model and the co-simulation model (Across network, p.u.).

In order to more accurately analyze the error between the co-simulation results and the original model results, Matlab's data simulation checker is still used here for further comparison. Figures 2.17 and 2.18 respectively show the comparison between the original and the co-simulation results for synchronization voltage and one of the inverter three-phase currents. As can be seen more clearly in Figure, the differences

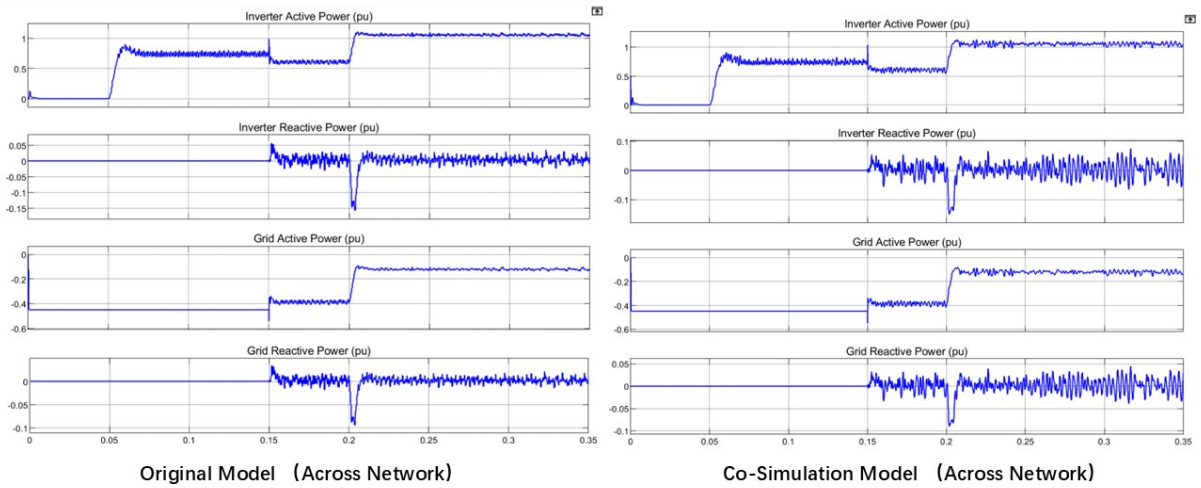

Figure 2.16: Comparison of inverter power and grid power between the original model and the co-simulation model (Across network, p.u.).

in the simulated signals are mainly centered on the locations of the peaks and troughs, as well as the intervals between 0 and 0.05 seconds where the signal frequency is higher. This is further evidence that the co-simulation interface still needs to be improved when dealing with rapidly changing signals. In Figure 2.19, the inverter power curves during the simulation process under local network and cross-network conditions are compared. It can be observed that the delay and packet loss in cross-network conditions have increased the average simulation error.

The rest of the results of the simulation data checker are appended to AppendixC, which includes the three-phase grid current/voltage signals abc, the three-phase inverter current/voltage signals abc, as well as the reactive and active power curves of the inverter and the grid. To facilitate comparison of the data, Table 2.2 was created to get a quick overview of the maximum error for each signal during the cross-network simulation, as well as the total error (in percent). The total error is calculated by summing the differences between every steps of the signals generated by the co-simulation and those generated by the original simulation model, and then calculating the percentage error.

Figure 2.17: Comparison of original model and co-simulation model of one phase of inverter three-phase current (Across network, p.u.).

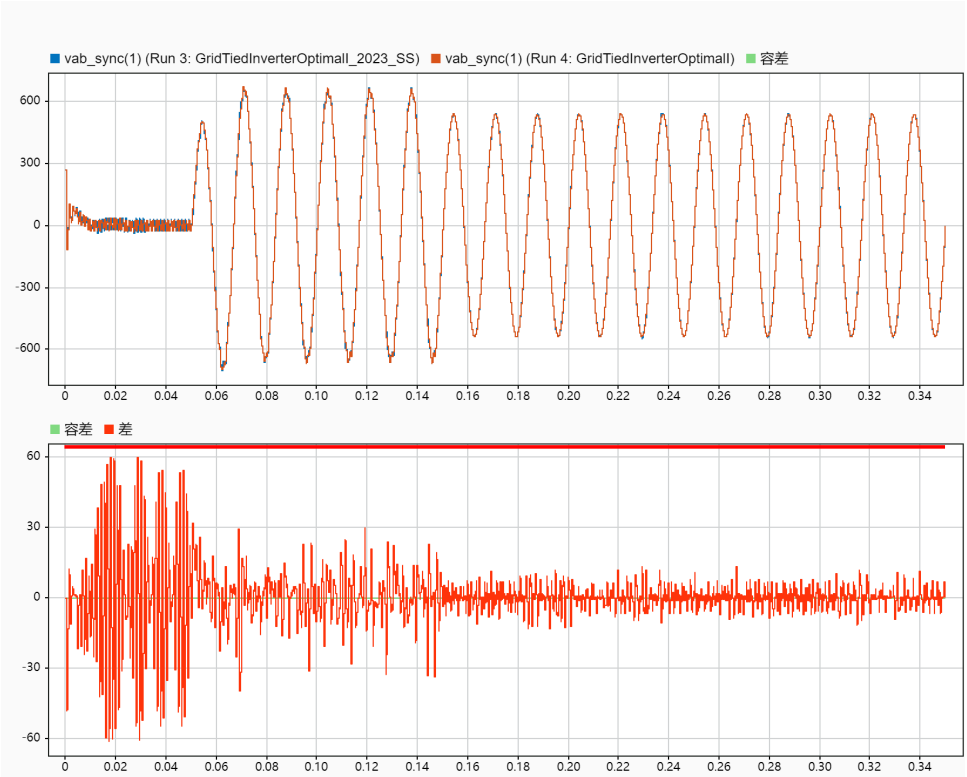

Figure 2.18: Comparison of original model and co-simulation model of inverter active power dynamic (Across network, V.).

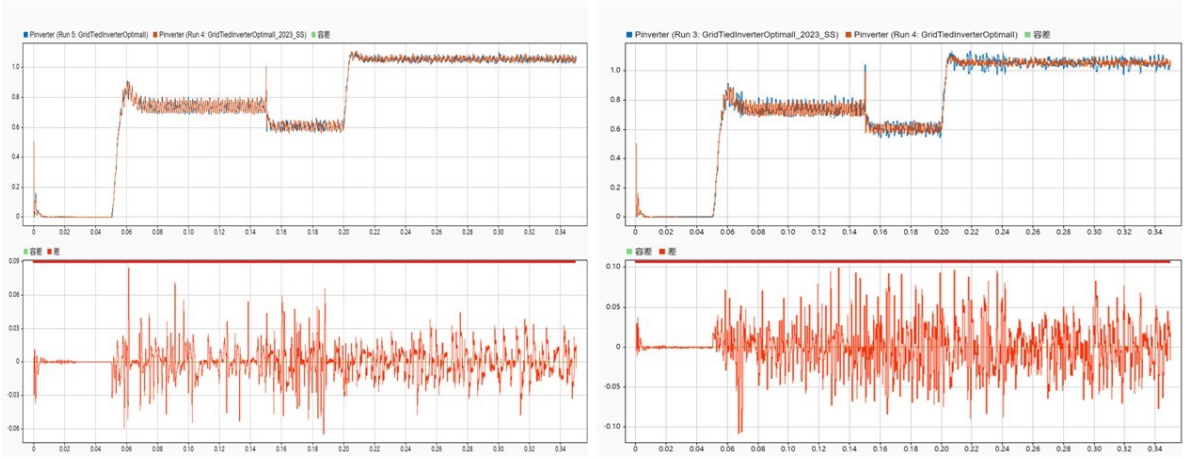

Figure 2.19: Comparison of inverter power dynamics (p.u.) within and across network conditions.

Signal Source	**Maximum Error(p.u.)**	**Total Error in Percentage(%)**
IabcInverter(a)	0.08	2.89
IabcInverter(b)	0.08	2.89
IabcInverter(c)	0.08	2.88
VabcInverter(a)	0.06	2.45
VabcInverter(b)	0.06	2.45
VabcInverter(c)	0.06	2.44
IabcGrid(a)	0.03	1.91
IabcGrid(b)	0.03	1.91
IabcGrid(c)	0.03	1.91
VabcGrid(a)	0.01	0.61
VabcGrid(b)	0.01	0.61
VabcGrid(c)	0.01	0.61
Pinverter	0.11	4.94
Qinverter	0.07	4.19
Pgrid	0.09	4.36
Qgrid	0.08	4.06

Table 2.2: Error analysis of each simulated signal compared to the original model results (Cross Network).

3

Co-simulation of multi-bus transmission line systems

Contents

3.1 116 Bus Transmission Line Model and Partitioning Strategy 29
3.1.1 Original Model Introduction 29
3.2 Simulation Process 32
3.2.1 Interface Communication between Sub-models 32
3.2.2 Parallel Simulation Interface 33
3.2.3 Sub-model Initialization and Startup 33
3.2.4 Simulation Workstation 33
3.3 Simulation Results 34
3.3.1 Co-simulation Results in LAN 34
3.3.2 Co-simulation Results Across Campus Network 37

3.1 116 Bus Transmission Line Model and Partitioning Strategy

3.1.1 Original Model Introduction

The project used a test model using a 116-bus, 28-power plant network for simulation purposes. This model is an extension of the 29-bus model from the Matlab case library [24]. Within this model, the load flow tool of Powergui was used to initialize a 735 kV transmission network across the 116 buses, and detailed modeling was performed for 28 power plants operating at 13.8 kV, with a total available generation capacity of 104,800 MVA. These include hydraulic turbines, governor systems, excitation systems, and power system stabilizers. The 735 kV transmission network is compensated using fixed

capacitors and inductors in both series and parallel configurations. Loads are connected to a 25 kV distribution system through 735 kV/230 kV and 230 kV/25 kV transformers. Additionally, the wind generation system is supported by a 6000 MW load (modeled with constant impedance and constant PQ) connected to the 120 kV bus. 9 MW wind farms, equipped with asynchronous generators, are linked to the 120 kV bus via 25 kV feeders and 25 kV/120 kV transformers. The model is discretized with a sampling time of 75 microseconds, and a six-cycle three-phase fault is programmed on the MTL bus. The model is a typical multi-bus transmission line model and provides relatively convenient decoupling conditions for easy interface settings. Figure 3.1 shows the model used for the simulation and briefly describes its modules function, while in Figure 3.2 the subsystems within the model are shown.

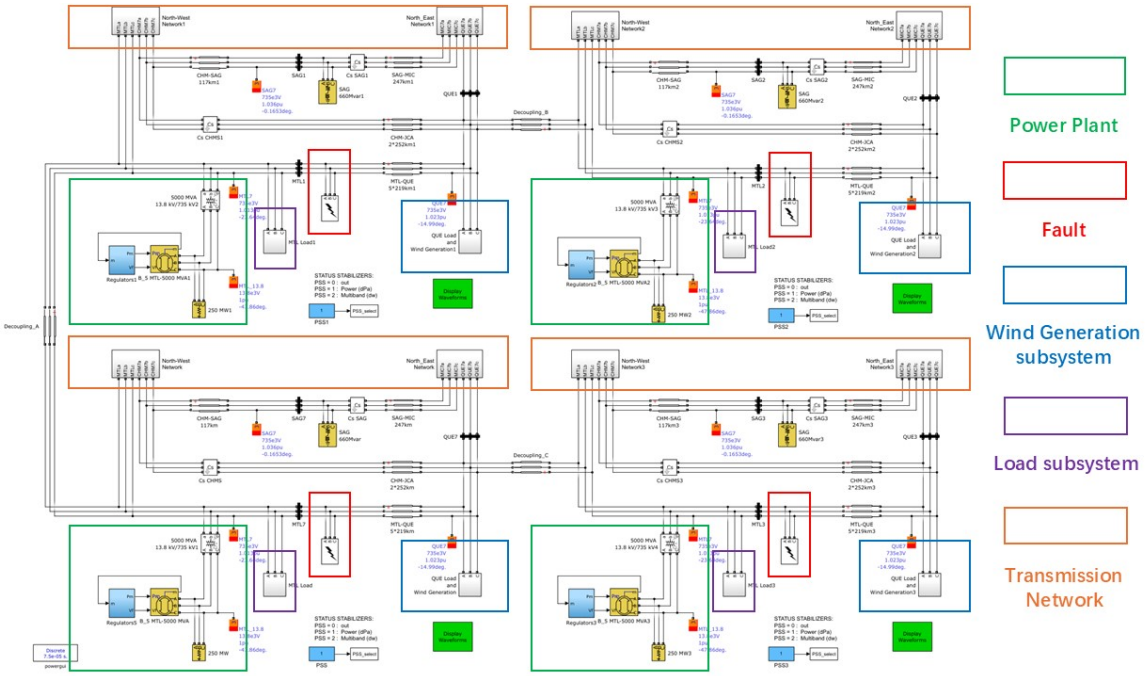

Figure 3.1: 116 bus transmission line system (original simulation model).

This 116bus model is actually a combination of the four original 29bus models, so we can follow the scope of the original model to observe the EM transient simulation of the large power grid. As Figs. 3.3 and 3.4 show the variations of the terminal voltage of the synchronous motor and the three-phase voltage of the load during the simulation. The simulation lasts for 4.5 seconds, during which a three-phase short-circuit fault is set up. In the simulation, this fault module will introduce a three-phase grounded short circuit at 0.2 seconds and the fault will clear at 0.3 seconds and the system will go through a recovery process. This can be seen in the sudden change in load voltage at 0.2-0.3 seconds, which also causes fluctuations in the motor speed. This fault setting is used to test the stability and responsiveness of the interface in unexpected situations during subsequent co-simulation.

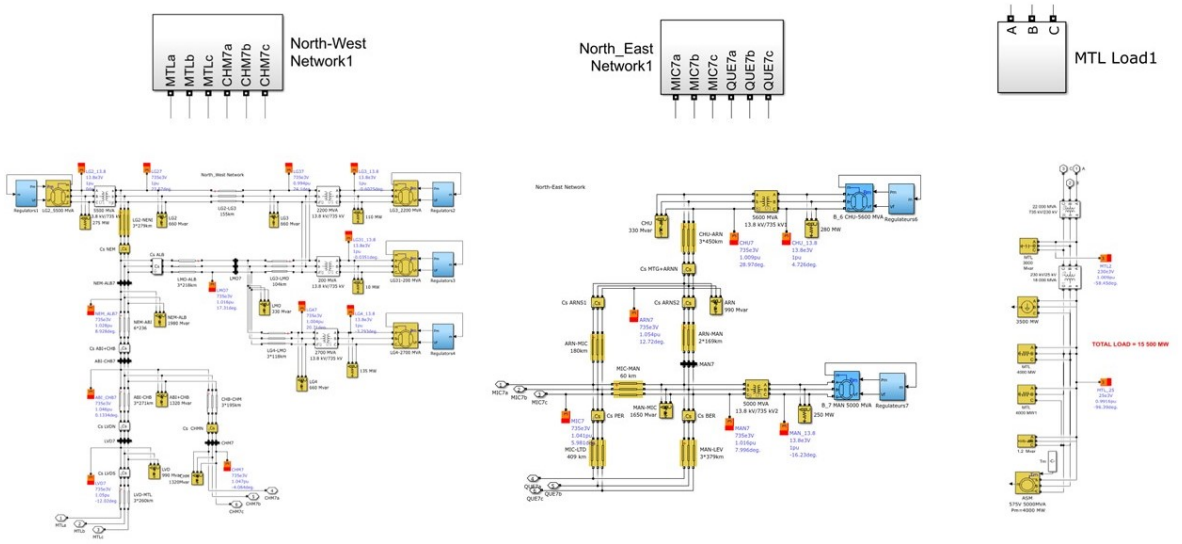

Figure 3.2: Subsystem expansion within the model.

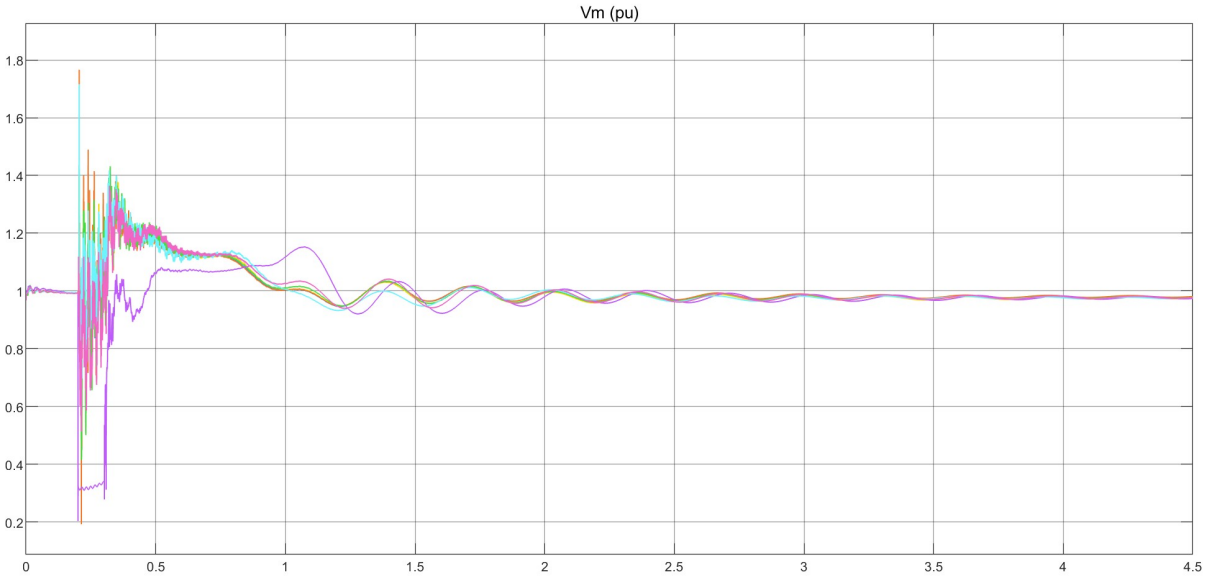

Figure 3.3: Synchronous motor terminal voltage (p.u.).

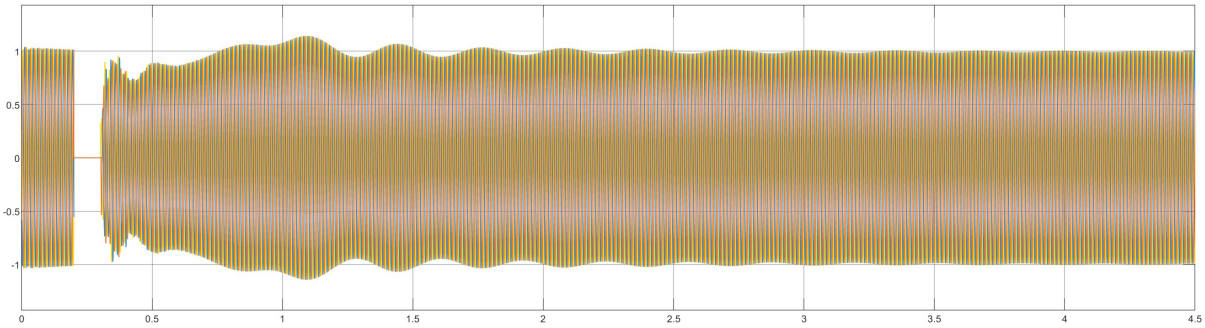

Figure 3.4: Load three-phase voltage (p.u.).

3.2 Simulation Process

3.2.1 Interface Communication between Sub-models

As shown in Figure 3.5, it decoupled the transmission lines to divide the original model (bottom left corner) into four sub-models. Communication between the sub-models is facilitated through interfaces. There are two types of interfaces: parallel simulation interfaces, which are used for communication between sub-models on the same computer (Sub1 and Sub4 are connected on computer 1, while Sub2 and Sub3 are connected on computer 2), and co-simulation interfaces, which are used for communication between models across computers, establishing the connection between Sub1 and Sub2.

The method of port decoupling has been explained in Chapter 1, and the code implementation logic for the co-simulation interface is described in Chapter 2. The original code can be found in Appendix A. It should be noted that in this model, the ports transmit the signals exchanged after the decoupling of the transmission lines. The number of parameters for these signals is 3, so the original 6 ports need to be reduced to 3. In this simulation, timestamp data has been removed because using timestamps among the four sub-models significantly increases the computational load of the interfaces, leading to longer simulation times. The Memory module is re-employed to prevent simulation deadlocks. Additionally, the sampling time used in the simulation must be consistent with the model step size (75 microseconds). The three-way handshake protocol for the TCP interface is still implemented using the `fopen` function.

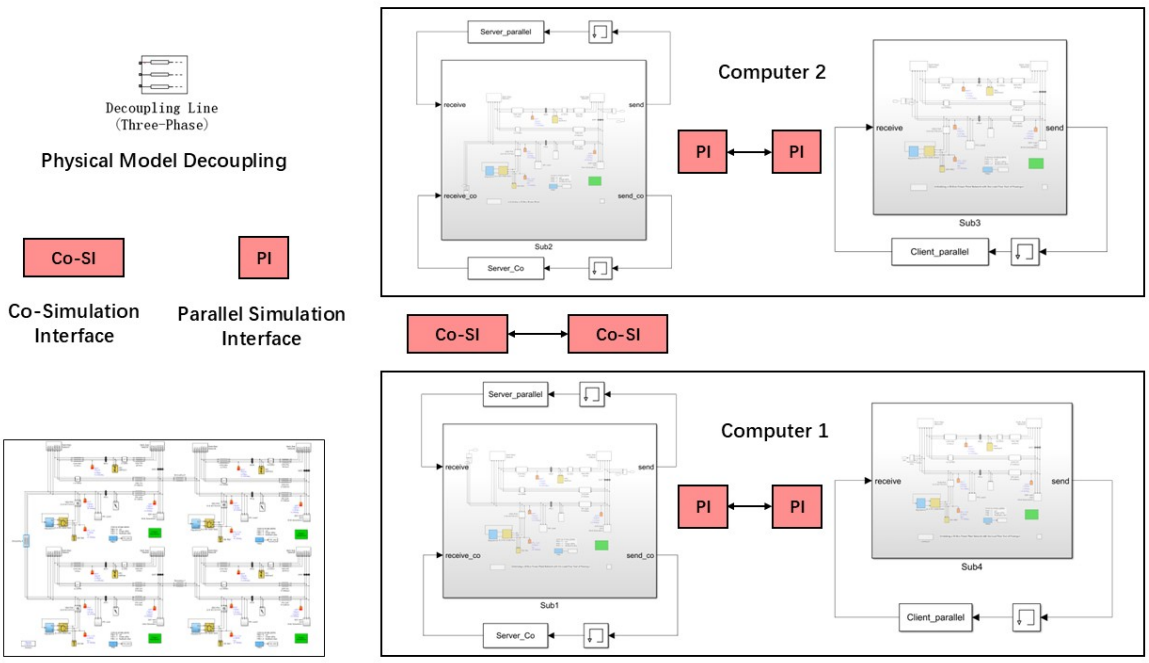

Figure 3.5: Interface communication between sub-models.

3.2.2 Parallel Simulation Interface

The Parallel Simulation Interface is divided into Server and Client ports and is also based on the Matlab S-Function module. The implementation logic for the parallel simulation interface is relatively simple. Since the simulation is conducted on the same computer, TCP/IP communication is no longer needed. Data exchange is achieved through shared memory using global variables. In the interface code, the global variables `sharedDataFromServer` and `sharedDataFromClient` are used to share data between different modules or threads, enabling efficient data exchange. The code for parallel simulation interface was attached in AppendixD.

3.2.3 Sub-model Initialization and Startup

In the simulation process, both co-simulation and parallel simulation are involved. Co-simulation ensures that the interface exchange signals are synchronized through a listening mechanism. However, parallel simulation interfaces lack such a synchronization design, which may lead to asynchronous signal exchanges between sub-models. Therefore, to ensure that all sub-models can start simultaneously and synchronize correctly, a `matlab.m` file must be manually created to start all sub-models. In this process, MATLAB's `parsim` function is used, which can efficiently run multiple sub-models in a parallel simulation environment. [25] `parsim` assigns each sub-model to different parallel workers, improving simulation efficiency while effectively utilizing multi-core processor resources. Additionally, since the model complexity between sub-models may differ, commands need to be added to ensure that all sub-models complete initialization before running the simulation. The specific startup code can be found in Appendix D. Figure 3.6 shows the operation of the parallel pool in the Matlab activity monitor.

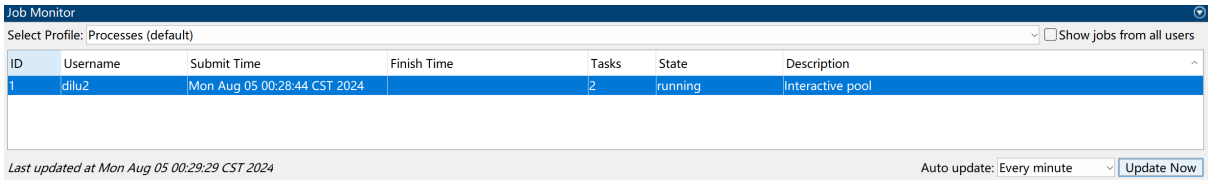

Figure 3.6: Simulation models running in parallel pools.

3.2.4 Simulation Workstation

Same as in Chapter2, the simulation will be run in both LAN and cross-network (apartments - Imperial College Campus) scenarios, and the same computer will be used as the simulation workstation. In this simulation, in addition to simulation errors, simulation duration is also an important metric. We use the

computer's built-in clock to measure the total time taken from pressing the start simulation button to the completion of the final simulation step, including the model initialization time.

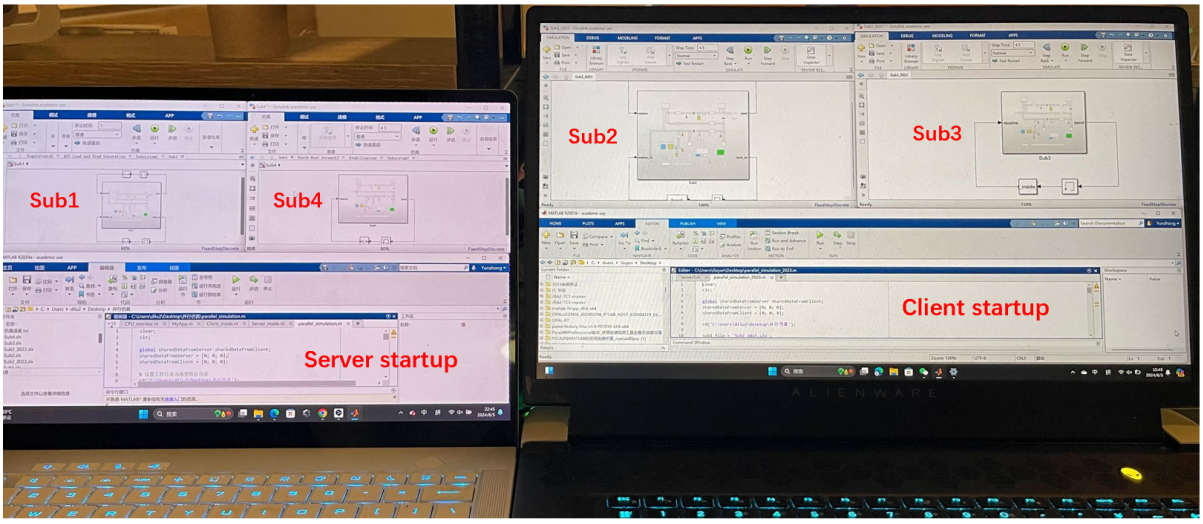

Figure 3.7: Simulation workstation construction.

3.3 Simulation Results

3.3.1 Co-simulation Results in LAN

Figures 3.8 and 3.9 show a comparison of the original model and co-simulation results in a local area network scenario, including the motor speed, synchronous motor terminal voltage, and load three-phase voltage. The recorded signals are from the motor located in the North-West Network1 sub-model. A three-phase ground fault occurs at 0.2 seconds in the simulation circuit and ends at 0.3 seconds. In this test, the precision of the parallel simulation interface is up to 9 decimal places, while the precision of the co-simulation interface is up to 7 decimal places. It can be seen that the co-simulation results are basically consistent with the original model simulation results.

To further investigate the errors, Simulation Data Inspector was used. The Scope is connected to a total of 14 signals, including 6 motor speed signals, 6 motor terminal voltage signals, 1 motor power factor signal, and 1 load three-phase voltage signal. We selected one motor's speed signal, terminal voltage signal, and the load's three-phase voltage signal as the subjects of our study, as shown in Figures 3.10, 3.11, and 3.12.

According to the plots, the simulation error of the co-simulation interface in the multi-bus transmission line system is much smaller than the simulation error of the inverter grid-connected system described in Chapter 3, even though the model complexity of the multi-bus transmission line is higher. Especially

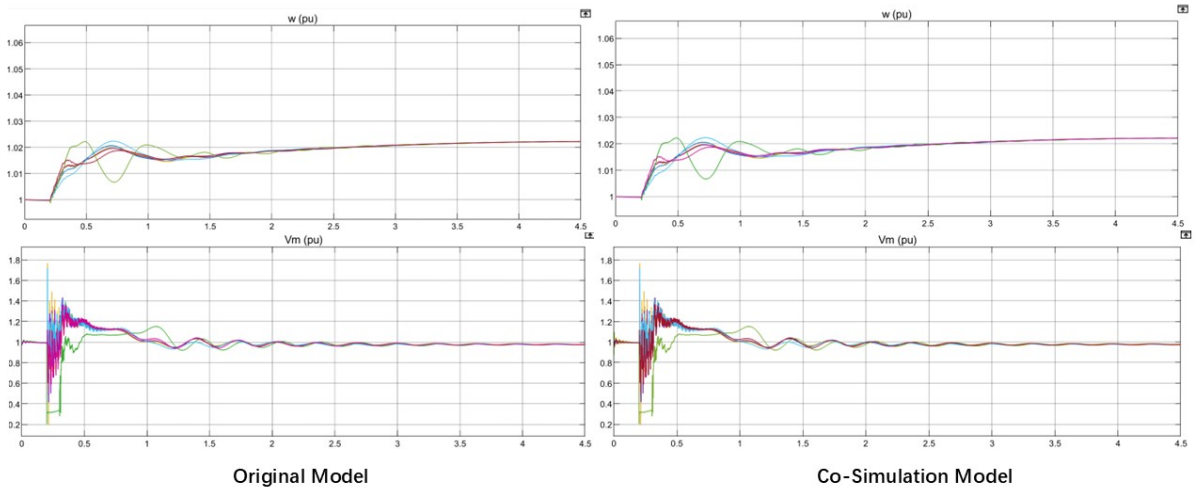

Figure 3.8: Simulation results of motor speed, synchronous motor terminal voltage (LAN).

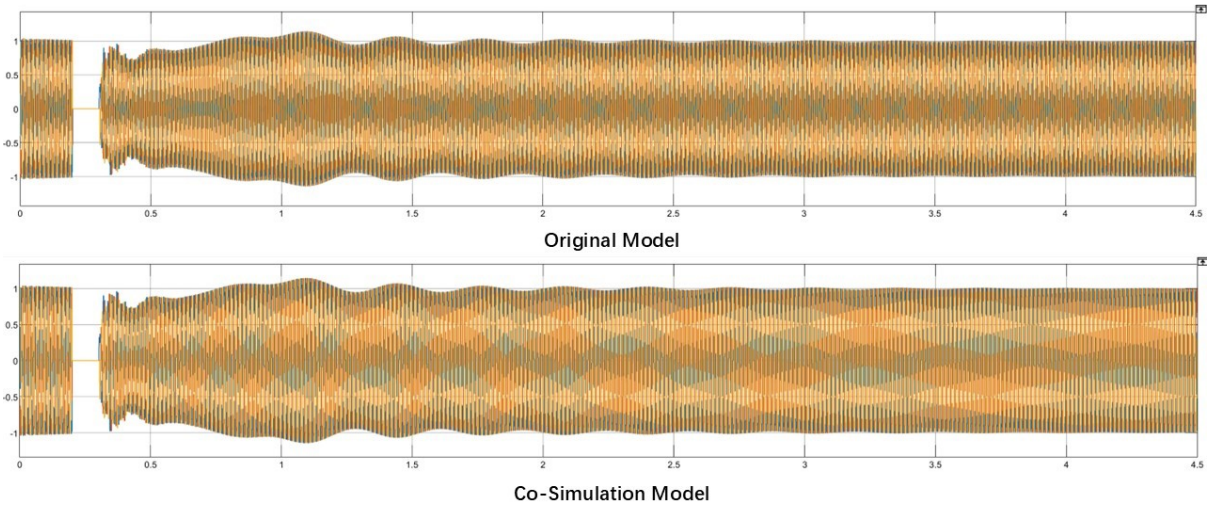

Figure 3.9: Simulation results of Load three-phase voltage signal (LAN).

in the motor terminal voltage curve, except for the errors caused by rapid numerical fluctuations due to the short circuit in the initial stage, the subsequent errors are almost negligible. This difference may be because, in the inverter grid-connected system, the controller is separated from the original model, and the interface exchanges control signals. Since the closed-loop PWM control signals in the system inherently have delays when controlling the current/voltage through actuators (the controller in Chapter 3 is not based on predictive control) and with fast frequency even slight variations in the control signals can more easily lead to errors in the controlled voltage or current. In contrast, in the multi-bus transmission line system, the interface only exchanges the decoupled three-phase voltage/current signals. As long as the transmitted data maintains sufficient precision and in a low-latency local area network environment,

errors are almost not introduced. However, due to the potential precision loss caused by decoupling and possible slight network delays, the interface still introduces minimal errors. Therefore, it can still be seen that in Figure 3.12, the load three-phase voltage curve that the small simulation errors gradually accumulate over time and increase as time progresses.

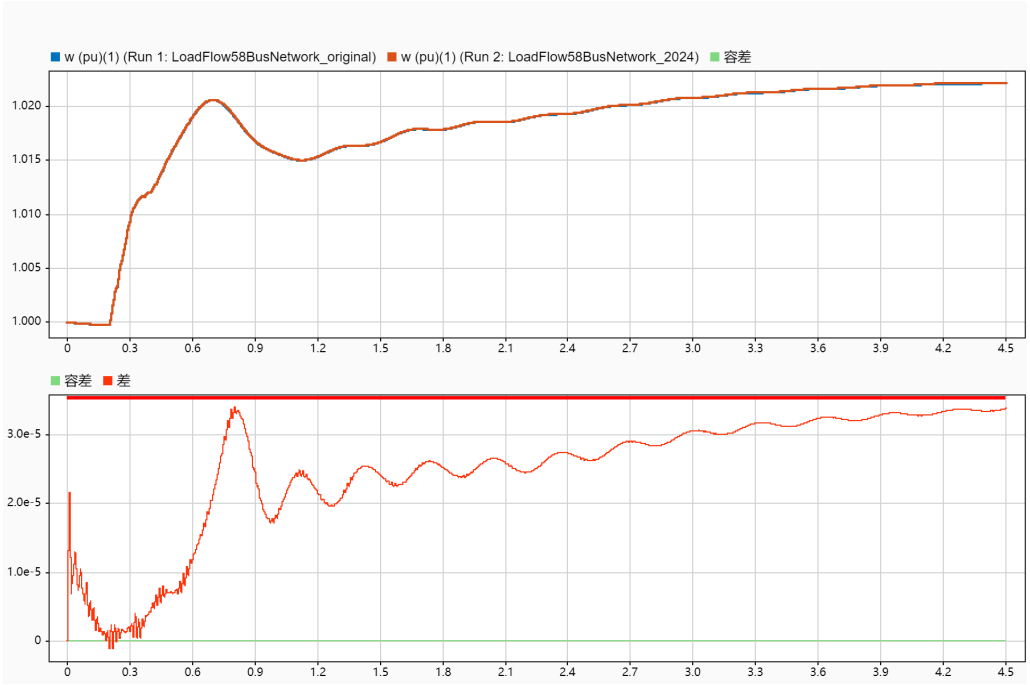

Figure 3.10: Comparison of motor speed curves between the original simulation model and the co-simulation model (LAN).

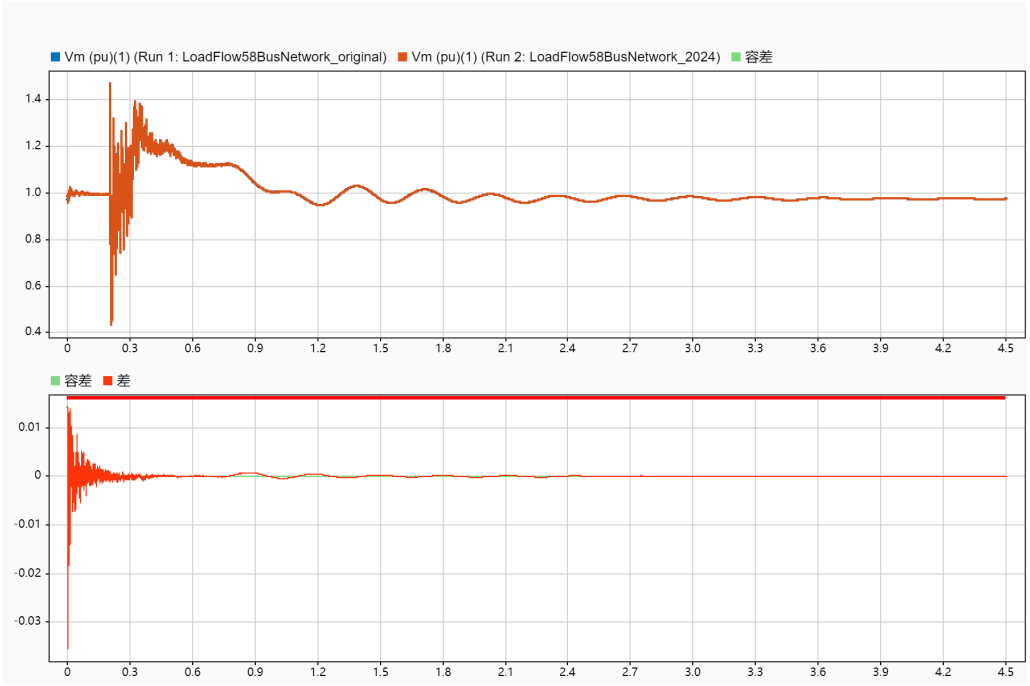

Figure 3.11: Comparison of synchronous motor terminal voltage curves between the original simulation model and the co-simulation model (LAN).

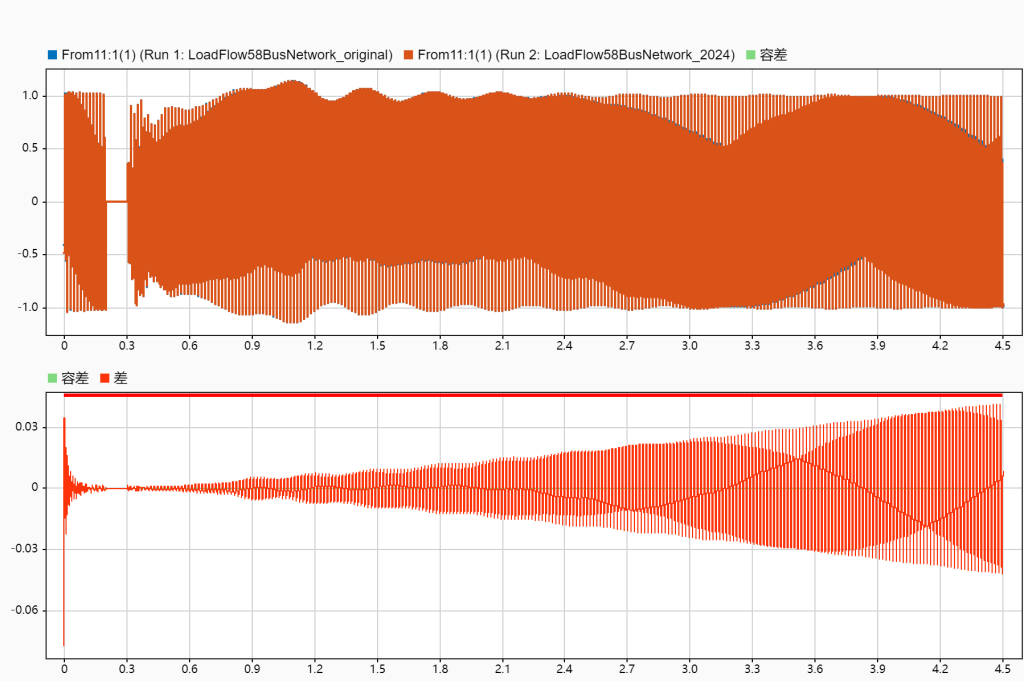

Figure 3.12: Comparison of load three-phase voltage (single-phase) curves between the original simulation model and the co-simulation model in a local area network (LAN) environment.

The results of the LAN simulation were not tabulated again to count the maximum and average errors for each curve because the errors were very small except for the three-phase voltage curves. The rest of the analyzed plots can be seen in AppendixE.

3.3.2 Co-simulation Results Across Campus Network

In cross-network co-simulation, delays and packet loss are unavoidable issues that primarily affect the accuracy and overall duration of the simulation. In addressing the issue of data packet loss, we adopt the strategy outlined in Chapter 3, which involves using the data from the previous packet to compensate for the lost packet. In this simulation, the interface transmits three-phase voltage and current signals rather than PWM control signals from the grid-connected inverter model. Consequently, this compensation strategy is more effective in this context, as voltage signals are inherently more stable compared to control signals, exhibiting less variation and slower frequency changes.

The testing process is consistent with previous procedures. Figures 3.13 and 3.14 compare the co-simulation results with those of the original model simulation, while Figures 3.15, 3.16, and 3.17 examine specific curves: a motor speed curve, a synchronous machine terminal voltage curve, and one phase of the three-phase voltage curve, respectively, using a simulation data checker for more detailed analysis.

As shown in Figures 3.15 and 3.16, the error at each step does not significantly increase due to cross-network delays and packet loss. The errors remain concentrated in regions where the voltage fluctuates

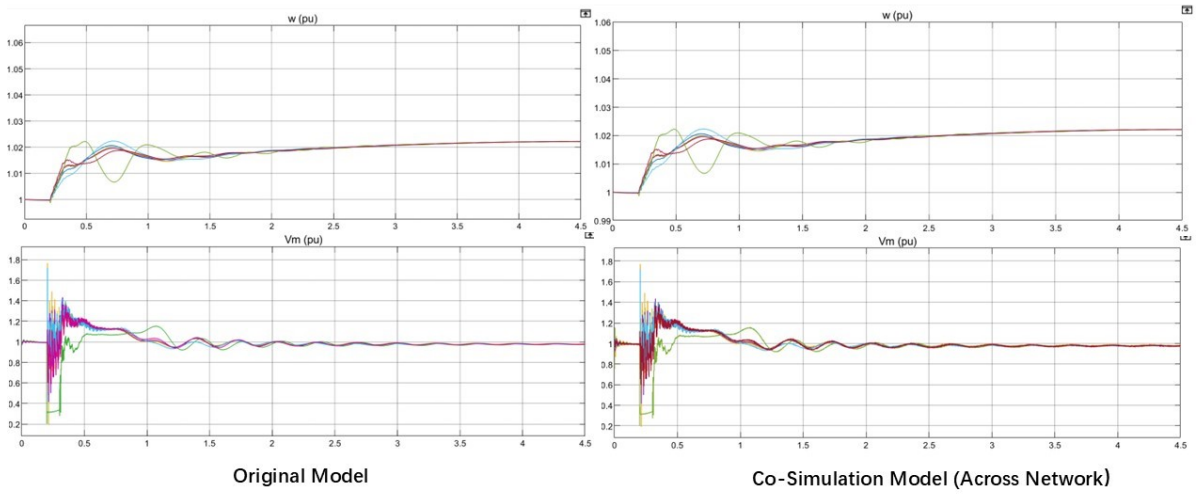

Figure 3.13: Simulation results of motor speed, synchronous motor terminal voltage (Across Network).

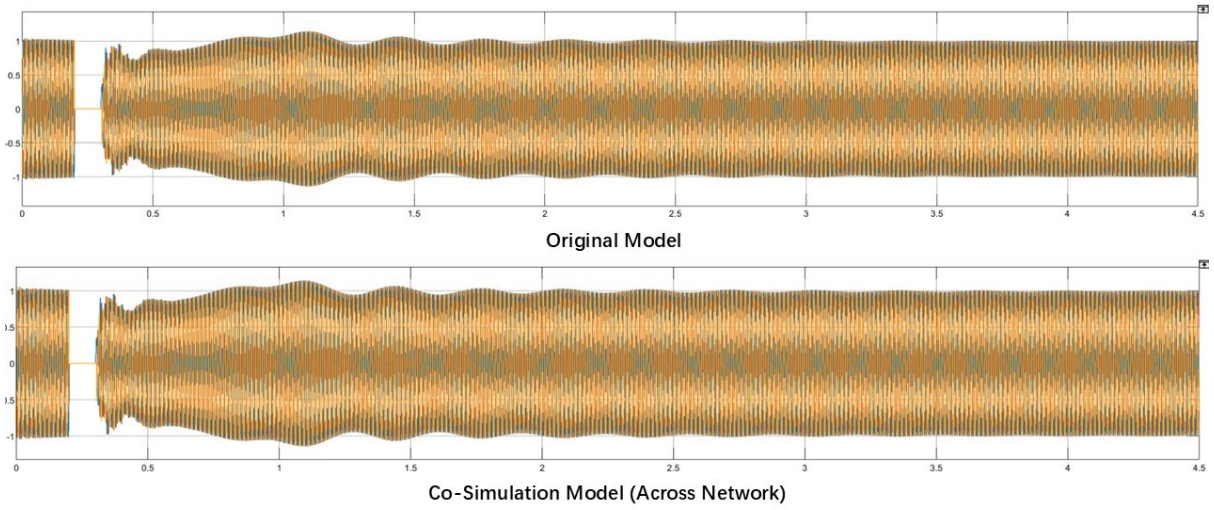

Figure 3.14: Simulation results of Load three-phase voltage signal (Across Netowrk).

rapidly, such as the peaks and troughs of the curves. As illustrated in Figure 5, after applying the packet loss compensation mechanism, the simulation error of the load voltage remains within 2% throughout the entire simulation process.

Although a similar compensation strategy was applied to the grid-connected inverter in Chapter 2, the results were notably different. This discrepancy is because the control loop of the inverter was effectively cut during the separation, leading to cumulative errors that caused the controller to gradually deviate from its control target. However, in this multi-bus transmission line model, only the transmission lines were decoupled, so the signal loss due to packet loss does not affect the simulation accuracy in subsequent steps.

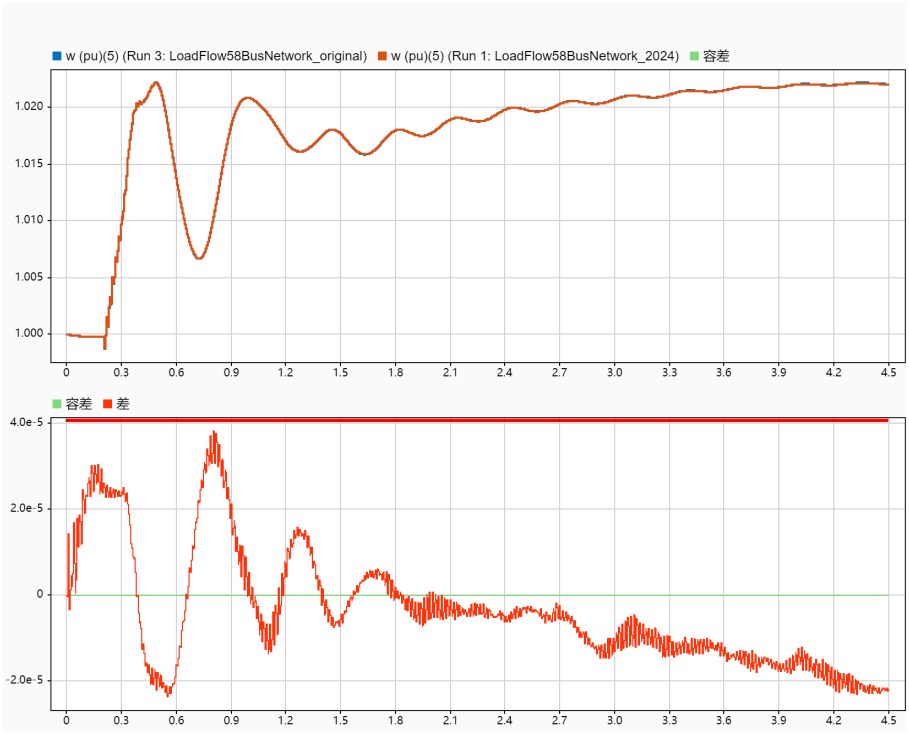

Figure 3.15: Comparison of motor speed simulation results between original model and cross-network co-simulation. (Top: Comparison of simulation curves; Bottom: Error values at each step)

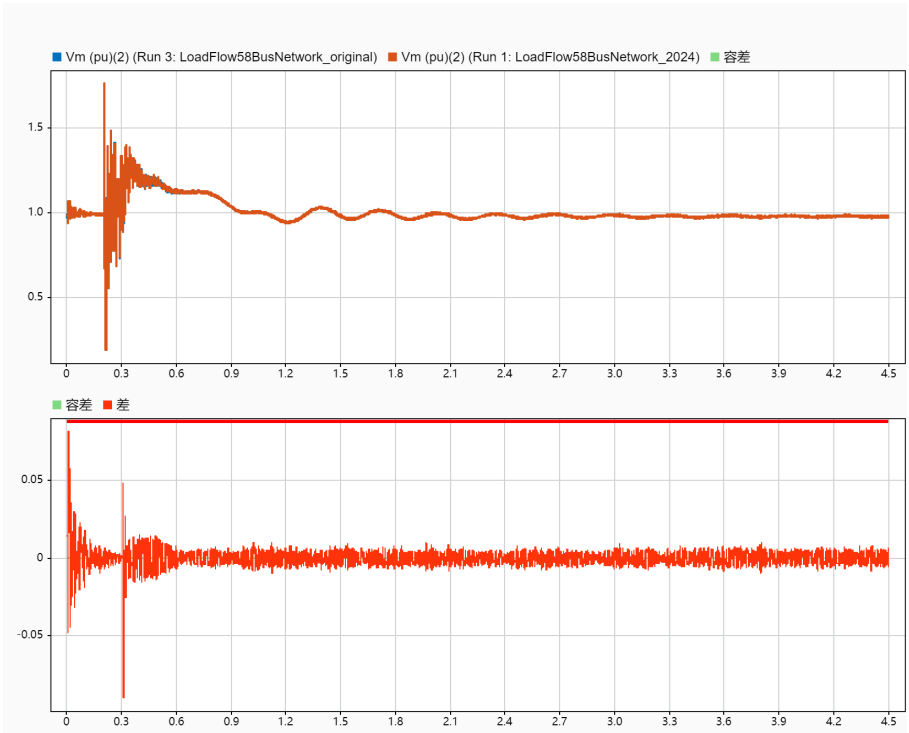

Figure 3.16: Comparison of synchronous motor end-voltage simulation results between original model and cross-network co-simulation. (Top: Comparison of simulation curves; Bottom: Error values at each step)

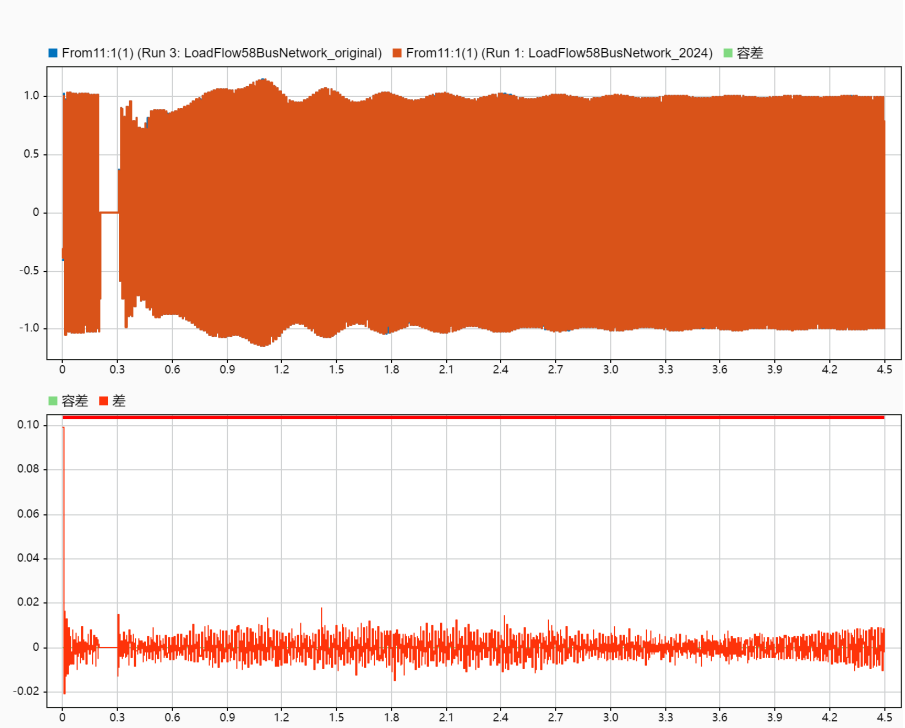

Figure 3.17: Comparison of load three-phase voltage simulation results between original model and cross-network co-simulation. (Top: Comparison of simulation curves; Bottom: Error values at each step)

A major drawback of the collaborative simulation method designed in this project is the extended simulation time. This issue primarily arises from the use of synchronous communication mode to align signal data between the two ports at each simulation step. Compared to UDP, this method sacrifices speed for higher simulation accuracy. As previously mentioned, to address this issue, parallel simulation techniques are employed to accelerate the simulation process. Figure 3.18 shows the initialization and simulation times of the 116-bus transmission line model under different simulation modes. In this test, the simulation scale of the model is 4*29 bus system (116 bus) the simulation step is 75us, and the total time of the model is set to 4.5s

It can be observed that, compared to the original model, the initialization time in the collaborative simulation mode is reduced, but the simulation time increases significantly. The reason is that after splitting the model (in this case, dividing the 116-bus into two 58-bus sub-models), the complexity of the model is reduced, leading to faster initialization. However, because the collaborative simulation ports use the TCP protocol in synchronous communication mode, each data transmission requires acknowledgment, resulting in cumulative network transmission time, which significantly increases the simulation time cost.

Using parallel simulation alone can greatly accelerate the simulation speed, with the simulation time being less than half of the original model's. This is because parallel simulation allows multiple cores of the computer to participate in the simulation tasks simultaneously. By distributing different

computational tasks to each core, it fully utilizes computing resources, thereby reducing the processing burden on individual cores and improving simulation efficiency. However, the initialization time for parallel simulation is slightly higher than that for collaborative simulation because it requires the basic setup of the parallel pool at startup. This includes launching multiple worker processes, allocating computing resources, and performing necessary configurations. Although these initialization steps add some time costs, they are fully offset during the subsequent simulation process.

However, combining collaborative simulation with parallel simulation did not significantly reduce simulation time as expected. This further confirms that the primary factor limiting the efficiency of collaborative simulation is still the transmission speed of the collaborative simulation interface, which is mainly affected by network latency and communication mode. Nevertheless, parallel simulation still provides some benefits, reducing simulation time by about 15% compared to using collaborative simulation alone. This improvement is likely due to parallel simulation breaking down the model into smaller sub-models, which reduces the complexity of each sub-model and simplifies the computational tasks.

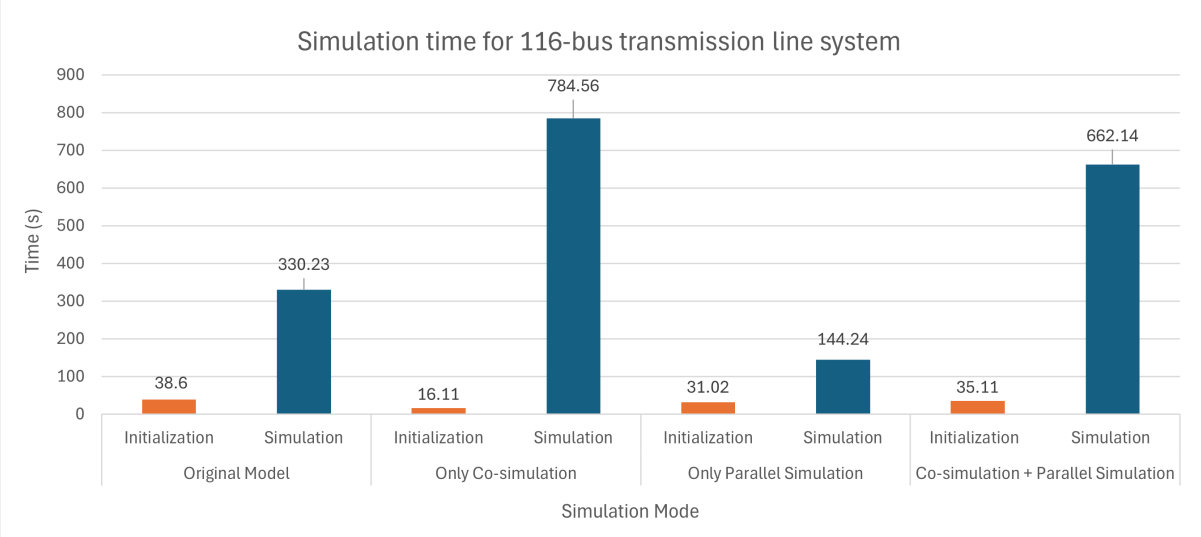

Figure 3.18: Comparison of simulation time for different modes.

4

Simulation Platform

Contents

4.0.1 Introduction 43

4.0.1 Introduction

Since the co-simulation interface is not a built-in component of Simulink but is established through Matlab Function, it requires a specific sequence of configurations before running, making the start-up and simulation process quite cumbersome. To simplify simulation operations and enhance the convenience of the co-simulation interface, the project plans to deliver a simulation platform as the final Deliverable, built using Matlab App. In this platform, users will be able to select models for parallel/co-simulation through the interface and view the simulation results.

As shown in Figure 4.1, the interface of the simulation platform App integrates various functionalities including initialization of the simulation program (to ensure concurrent simulation runs), model selection, simulation start, result clearing, simulation time setting, simulation status display, and simulation result display. From the loaded models window, users can add the desired models to the selection area on the right using the 'Add Model' button, or remove models using the 'Remove' button. Once the simulation is running, the status display at the bottom will show messages such as "Simulation running for 'MODEL' and 'MODEL'" and "Simulation completed" to indicate the current status. Users can then view the desired simulation graphs from the simulation results section on the right. As shown in Figure 4.2, the resultant plot can be generated by calling the simulation results.

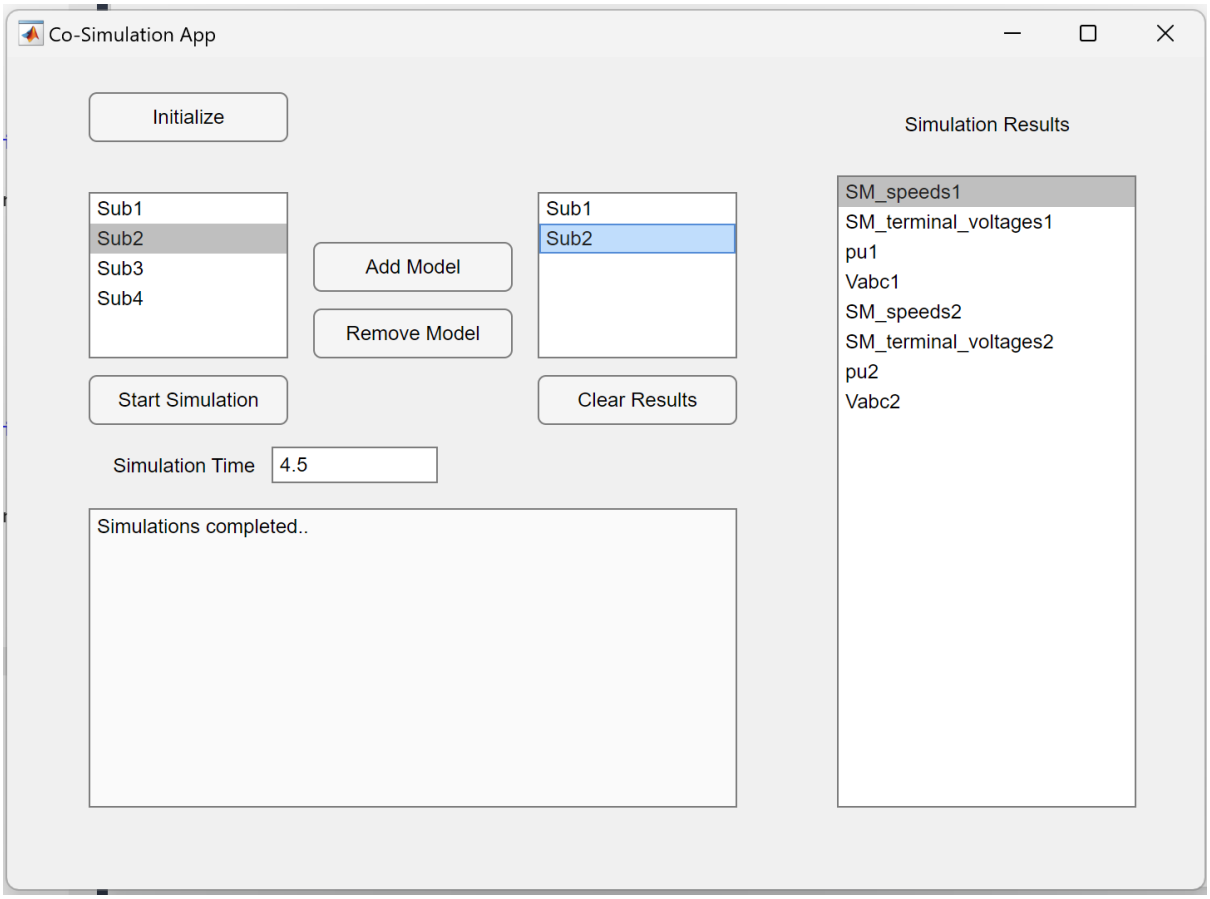

Figure 4.1: Simulation platform.

To ensure the smooth operation of the simulation platform, several preliminary settings must be completed. First, all sub-models for the simulation need to be placed in the same folder within the code directory path, including the .m files for the interface. Then, the names of the sub-models must be added to the app.ModelListBox.Items function so that the simulation models will appear in the selection box during initialization. Secondly, we need to set up the model result outputs. In Simulink, all signals recorded using Scope must be logged out to the workspace. This allows the simOut1.get function to fetch the data and display it in the graph on the right. Correspondingly, we need to change the names of the output signals in the callback functions within the App program to ensure the program runs correctly.

The platform can be used to simulate parallel simulations in a single computer. Note that before running, it is necessary to make sure that the interfaces of all sub-models have been set up. Additionally, to achieve co-simulation, the simulation platform program needs to be opened on both computers. During co-simulation, the interface on the Server side will remain in a continuous listening state until it receives a signal from the Client. Therefore, when running the simulation, click the ”Start Simulation” button on the Server side computer first, and then start the simulation on the Client side. The code for the simulation platform implemented in this project is provided in AppendixF.

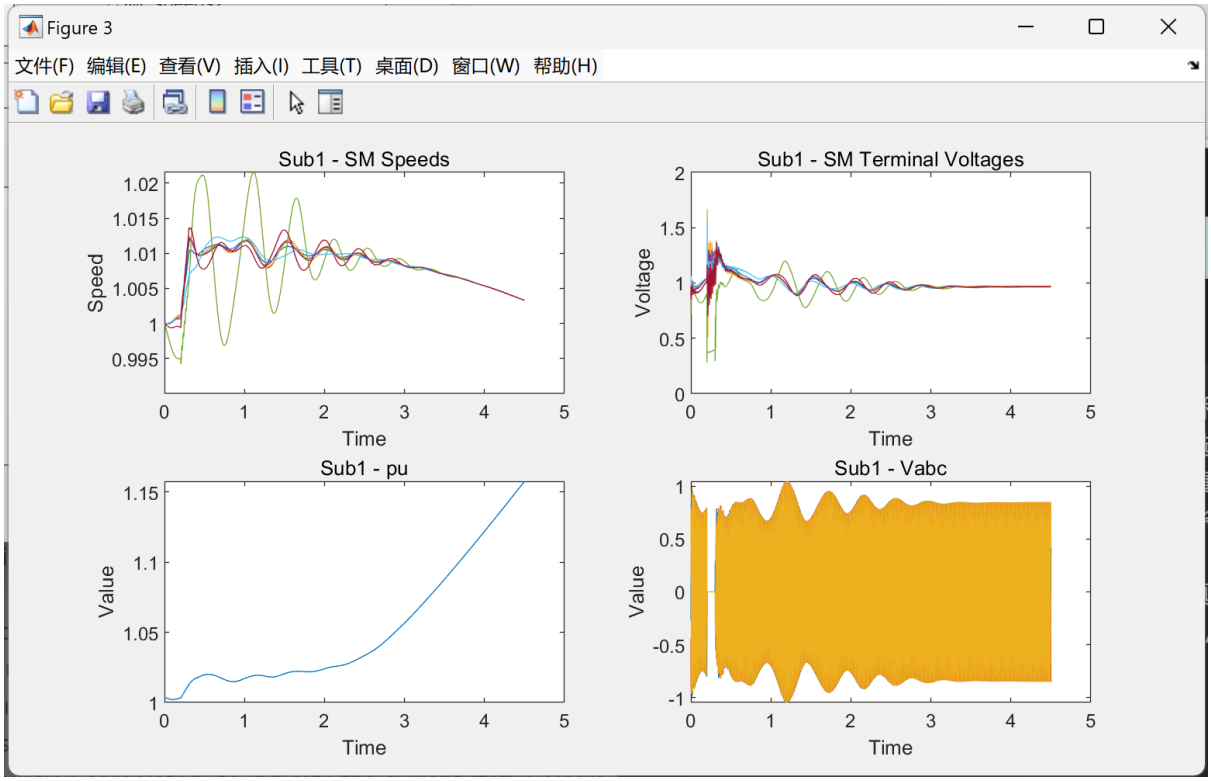

Figure 4.2: Exporting results from the simulation platform.

4

Conclusions

4.1 Conclusions

In this project, we designed a communication interface for co-simulation based on Matlab/Simulink. This interface uses the TCP/IP protocol, enabling signal transmission over both local area networks and wide-area networks. This allows separate simulation models to be connected through the interface, facilitating joint simulation among multiple parties without disclosing proprietary models, thereby protecting the models. By utilizing timestamps and synchronized communication mechanisms, this interface can reduce the impact of delays and packet loss in cross-network transmissions, thus achieving high stability.

The interface was tested with a grid-connected inverter model and a 116-bus transmission line model, and simulation results were recorded for both local area network and cross-network communication. The results show that the co-simulation in the local area network is consistent with the original model, while the cross-network co-simulation is mainly affected by packet loss, causing the simulation signals to deviate over time but still remaining within a 5% error range. Additionally, the decoupling points when separating the original model have some impact on the simulation results. Decoupling at the transmission line yielded better results compared to decoupling at the inverter controller.

To improve simulation efficiency, the project also explored the combination of parallel simulation with co-simulation. Tests showed that running models in parallel can further enhance simulation speed, but the overall speed is still limited by the high latency accumulated from the synchronized communication mechanism. Parallel simulation can only slightly improve simulation speed by further reducing the complexity of the sub-models.

The project also established a simulation platform based on Matlab App as a deliverable for quickly implementing co-simulation. Using this platform, users can conveniently select objects for co-simulation or parallel simulation and avoid data misalignment caused by asynchronous starts. Through the simulation platform, users can quickly plot the desired simulation results, and conveniently find simulation results of interest.

4.2 Future Work

The synchronous communication protocol based on TCP/IP greatly limits the speed of this co-simulation method, thereby reducing the practicality of this simulation approach. In future experiments, it may be worth considering the use of the UDP protocol. By adding timestamp information to each data packet and adding interface algorithmic logic to reorder the data packets based on their timestamps at the receiving end, multiple data packets can be sent quickly.

Another drawback of this co-simulation is that the interface is a pure data interface, meaning it can only decouple the original model at specific points. Otherwise, the separated sub-models cannot run independently, such as in ring network structures. Additionally, the compatibility of the interface can be expanded. Currently, the interface implemented in this project can only be used in Matlab. Consideration could be given to creating the interface as an FMI (Functional Mock-up Interface) to enable communication between different simulation software.

A

A

Title of the Appendix

Appendix A: S-Function Code for Grid-connected inverter model Co-Simulation

```
function [sys, x0, str, ts] = Server3(t, x, u, flag)
switch flag
    case 0
        [sys, x0, str, ts] = mdlInitializeSizes();
    case 3
        sys = mdlOutputs(t, x, u);
    case {1, 2, 4, 9}
        sys = [];
    otherwise
        error(['Unhandled flag = ', num2str(flag)]);
end
end

function [sys, x0, str, ts] = mdlInitializeSizes()
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 6; % 6 output ports for control signals
sizes.NumInputs      = 6; % 6 input ports for voltage and current signals
sizes.DirFeedthrough = 1;
```

```
sizes.NumSampleTimes = 1;

sys = simsizes(sizes);
x0 = [];
str = [];
ts = [0.00001 0]; % Sample time

disp('Initializing TCP server...');

global t_server_receive t_server_send;
t_server_receive = tcpip('0.0.0.0', 30001, 'NetworkRole', 'server');
t_server_send = tcpip('0.0.0.0', 30000, 'NetworkRole', 'server');

set(t_server_receive, 'InputBufferSize', 1000000); % Buffer size
set(t_server_send, 'OutputBufferSize', 1000000);

set(t_server_receive, 'Timeout', 10); % Timeout 10 seconds
set(t_server_send, 'Timeout', 10);

fopen(t_server_receive);
fopen(t_server_send);
disp('Server initialized successfully.');
end

function sys = mdlOutputs(t, x, u)
global t_server_receive t_server_send;
persistent last_valid_control_signals;
if isempty(last_valid_control_signals)
    last_valid_control_signals = [0; 0; 0; 0; 0; 0]; % Initial control signals
end

% Package and send voltage and current data
voltage_current_data = [u(1:6)]; % Send voltage and current directly without
    identifiers
try
    fwrite(t_server_send, voltage_current_data, 'double');
    disp(['Voltage and current data sent to client: ' num2str(u(1:6)')]);
catch ME
```

```
58      disp(['Error during sending data: ' ME.message']);
59      % Attempt to reconnect
60      try
61          fclose(t_server_send);
62          fopen(t_server_send);
63      catch
64          disp('Failed to reconnect.');
65      end
66  end
67
68  % Receive control signals
69  try
70      while t_server_receive.BytesAvailable < 6 * 8 % Each time receiving 6 double
            precision numbers (control signals)
71          pause(0.0001);
72      end
73      control_signals = fread(t_server_receive, 6, 'double');
74      last_valid_control_signals = control_signals;
75      disp(['Control signals received from client: ' num2str(control_signals')]);
76  catch ME
77      disp(['Error during receiving data: ' ME.message']);
78      % Attempt to reconnect
79      try
80          fclose(t_server_receive);
81          fopen(t_server_receive);
82      catch
83          disp('Failed to reconnect.');
84      end
85  end
86
87  sys = last_valid_control_signals;
88
89  if any(isnan(sys))
90      sys = [0; 0; 0; 0; 0; 0];
91  end
92  end
```

Listing A.1: Server S-Function Code for Grid-connected inverter model

A

A

```
function [sys, x0, str, ts] = Client3(t, x, u, flag)
switch flag
    case 0
        [sys, x0, str, ts] = mdlInitializeSizes();
    case 3
        sys = mdlOutputs(t, x, u);
    case {1, 2, 4, 9}
        sys = [];
    otherwise
        error(['Unhandled flag = ', num2str(flag)]);
end
end

function [sys, x0, str, ts] = mdlInitializeSizes()
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 6; % 6 output ports for voltage, current, and control
    signals
sizes.NumInputs      = 6; % 6 input ports for voltage, current, and control signals
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;

sys = simsizes(sizes);
x0 = [];
str = [];
ts = [0.00001 0]; % Sample time

disp('Initializing TCP client send and receive...');

global t_client_send t_client_receive;
t_client_send = tcpip('192.168.56.1', 30001, 'NetworkRole', 'client');
set(t_client_send, 'Timeout', 10); % Timeout 10 seconds
set(t_client_send, 'OutputBufferSize', 1000000); % Increase output buffer size

t_client_receive = tcpip('192.168.56.1', 30000, 'NetworkRole', 'client');
set(t_client_receive, 'Timeout', 10); % Timeout 10 seconds
```

```
set(t_client_receive, 'InputBufferSize', 1000000); % Increase input buffer size

try
    fopen(t_client_send);
    disp('Client send initialized successfully.');
catch ME
    disp(['Failed to initialize client send: ' ME.message']);
end

try
    fopen(t_client_receive);
    disp('Client receive initialized successfully.');
catch ME
    disp(['Failed to initialize client receive: ' ME.message']);
end
end

function sys = mdlOutputs(t, x, u)
global t_client_send t_client_receive;
persistent last_valid_voltage_data last_valid_current_data;
if isempty(last_valid_voltage_data)
    last_valid_voltage_data = [0; 0; 0];
end
if isempty(last_valid_current_data)
    last_valid_current_data = [0; 0; 0];
end

% Package and send control signals
control_signals = [u(1:6)]; % Send control signals directly without identifiers
try
    fwrite(t_client_send, control_signals, 'double');
    disp(['Control signals sent to server: ' num2str(u(1:6)')]);
catch ME
    disp(['Error during sending control signals: ' ME.message']);
    % Attempt to reconnect
    try
        fclose(t_client_send);
        fopen(t_client_send);
```

```
    catch
        disp('Failed to reconnect.');
    end
end

% Receive voltage and current data
try
    while t_client_receive.BytesAvailable < 6 * 8 % Each time receiving 6 double
        precision numbers (voltage and current)
        pause(0.0001);
    end
    voltage_current_data = fread(t_client_receive, 6, 'double');
    voltage_data_recv = voltage_current_data(1:3);
    current_data_recv = voltage_current_data(4:6);
    last_valid_voltage_data = voltage_data_recv;
    last_valid_current_data = current_data_recv;
    disp(['Voltage data received from server: ' num2str(voltage_data_recv')]);
    disp(['Current data received from server: ' num2str(current_data_recv')]);
catch ME
    disp(['Error during receiving data: ' ME.message']);
    % Attempt to reconnect
    try
        fclose(t_client_receive);
        fopen(t_client_receive);
    catch
        disp('Failed to reconnect.');
    end
end

sys = [last_valid_voltage_data; last_valid_current_data];

if any(isnan(sys))
    sys = [0; 0; 0; 0; 0; 0];
end
end
```

Listing A.2: Client S-Function Code for Grid-connected inverter model

B

Title of the Appendix

Appendix B: Error comparison image between original model simulation results and co-simulation (Inverter grid-connected model)

Figure B.1: Active power.

Figure B.2: Reactive power.

Figure B.3: Inverter three-phase current A.

Figure B.4: Inverter three-phase current B.

Figure B.5: Inverter three-phase current C.

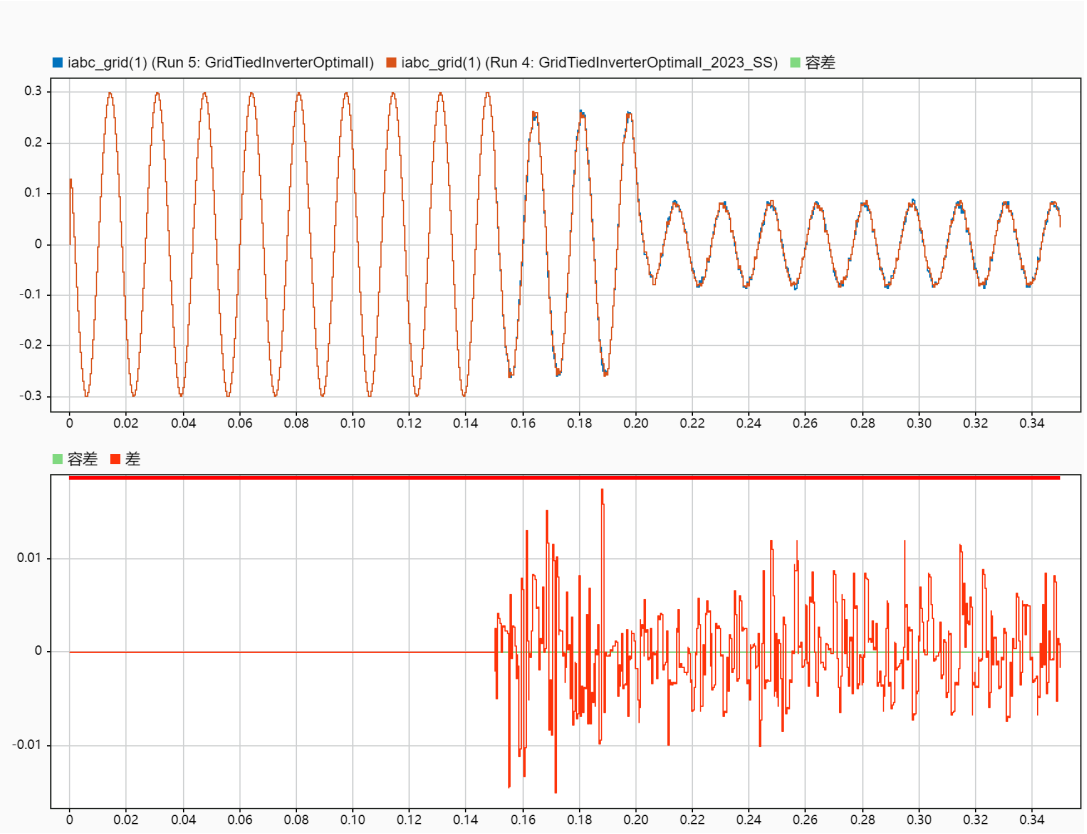

Figure B.6: Grid three-phase current A.

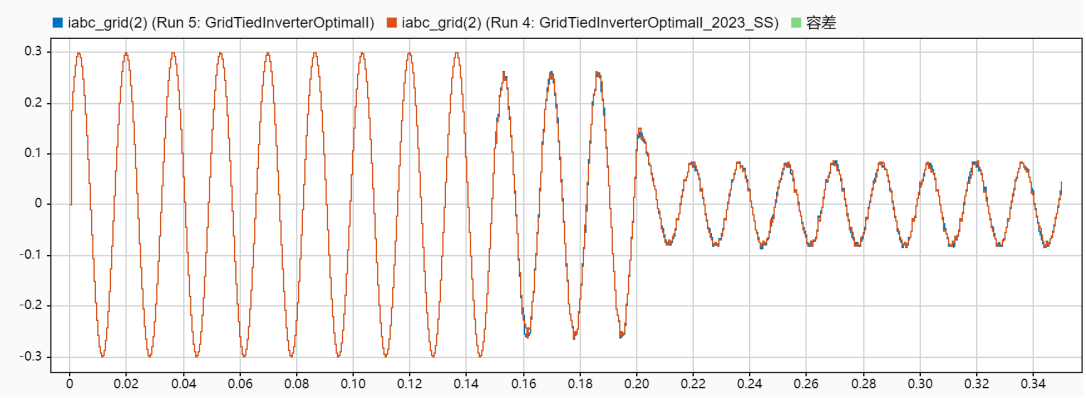

Figure B.7: Grid three-phase current B.

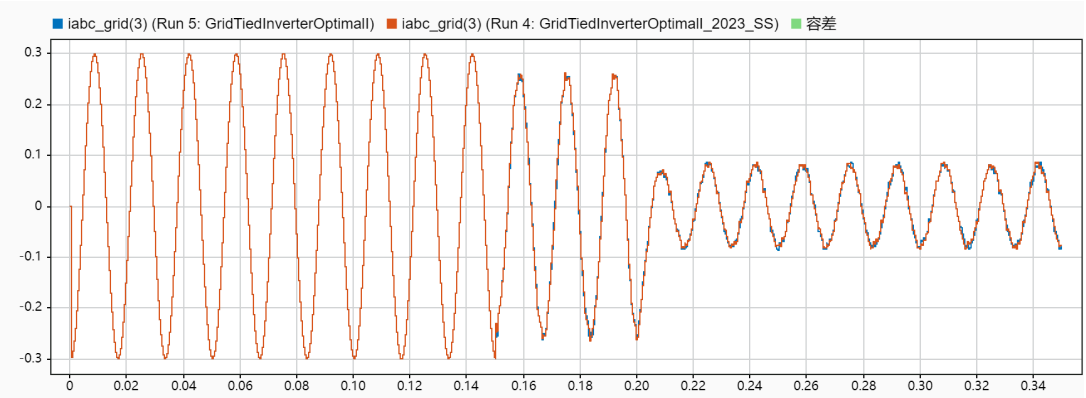

Figure B.8: Grid three-phase current C.

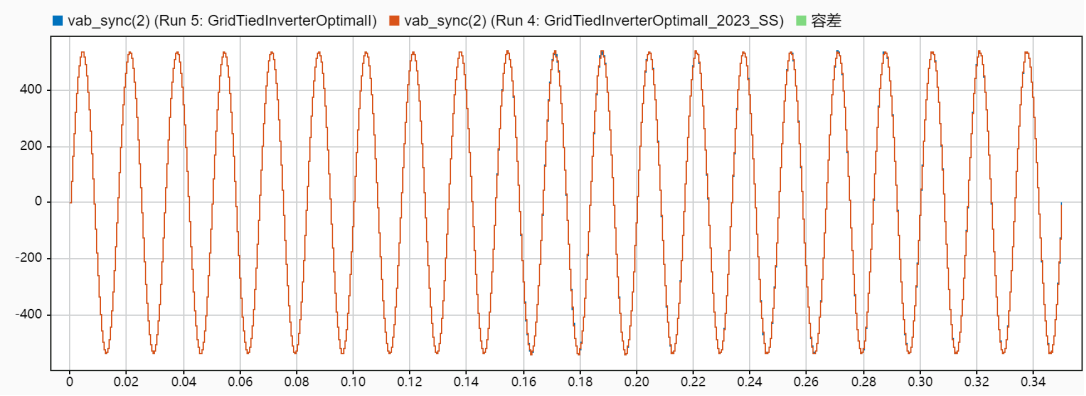

Figure B.9: Synchronous voltage.

B

C

Title of the Appendix

Figure C.1: Active power comparison (across networks).

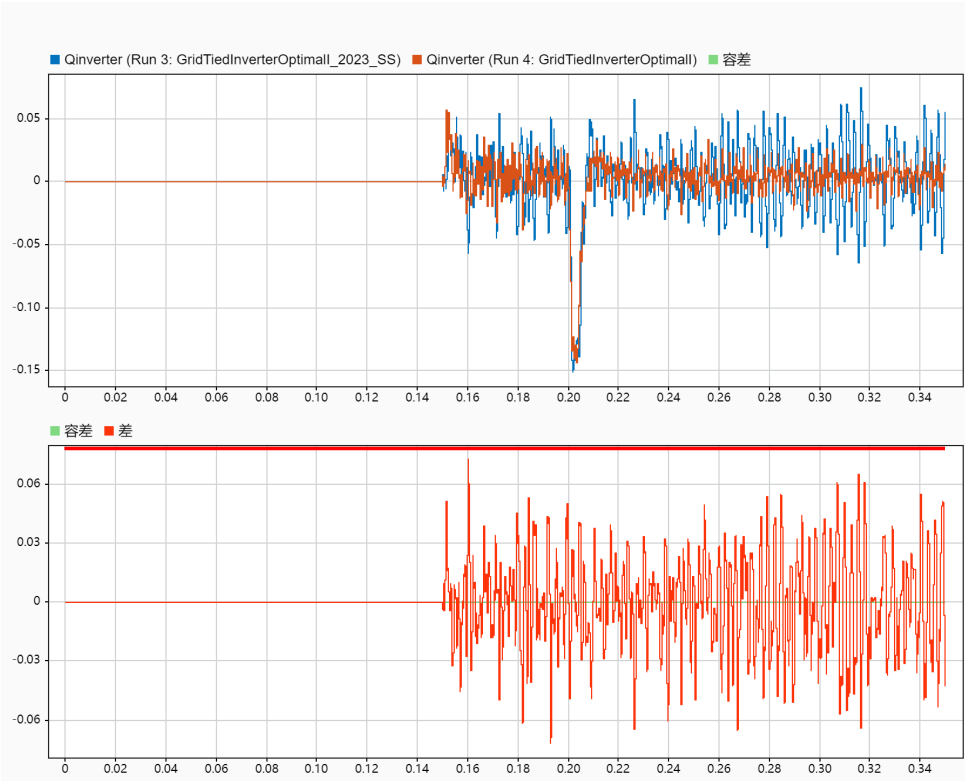

Figure C.2: Inactive power comparison (across networks).

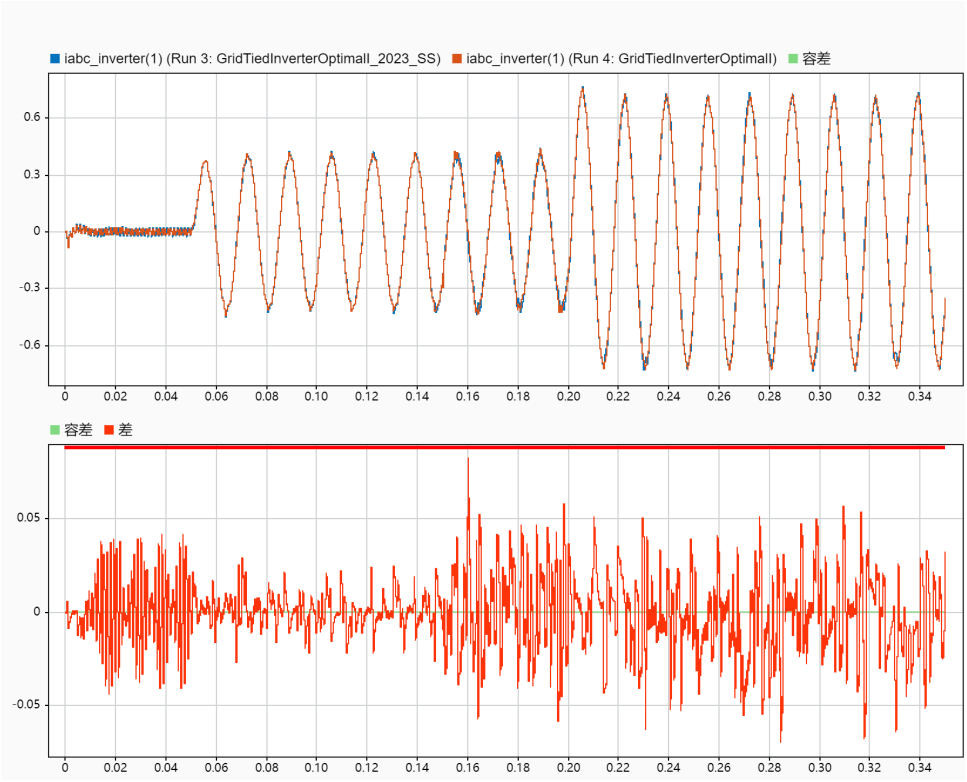

Figure C.3: Comparison of three phase inverter current a (across networks).

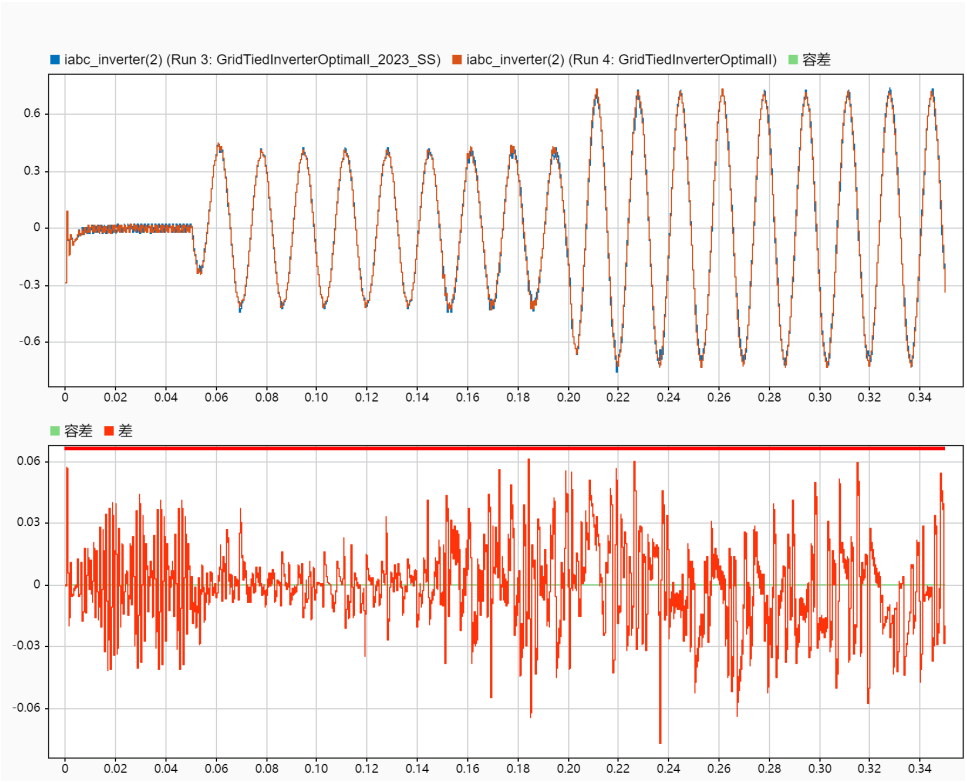

Figure C.4: Comparison of three phase inverter current b (across networks).

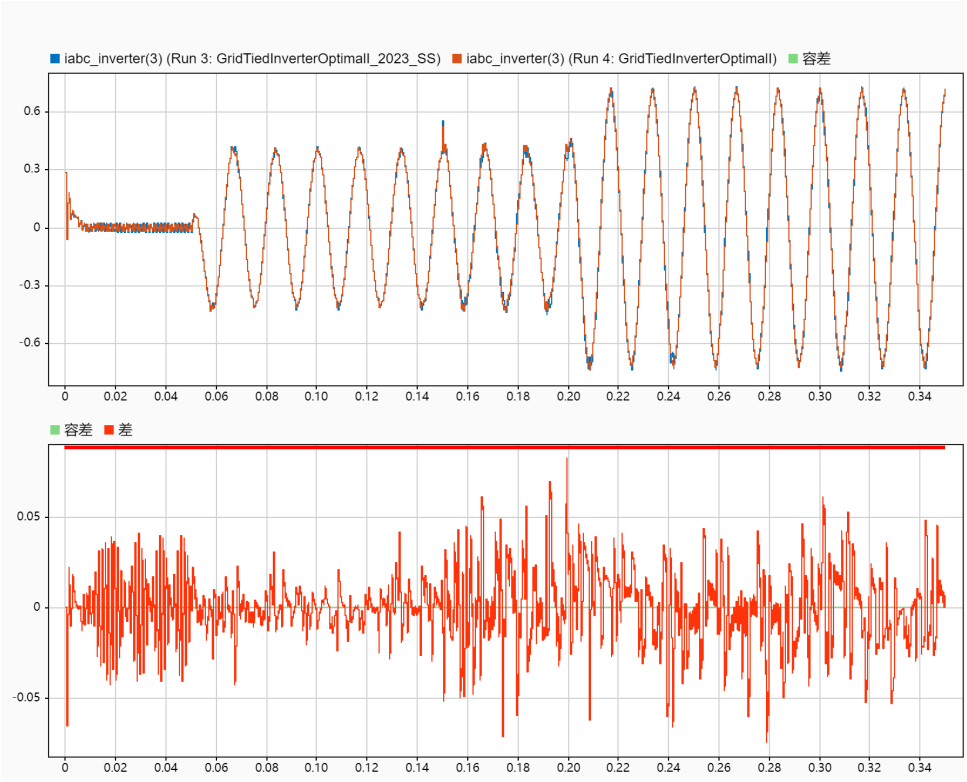

Figure C.5: Comparison of three phase inverter current c (across networks).

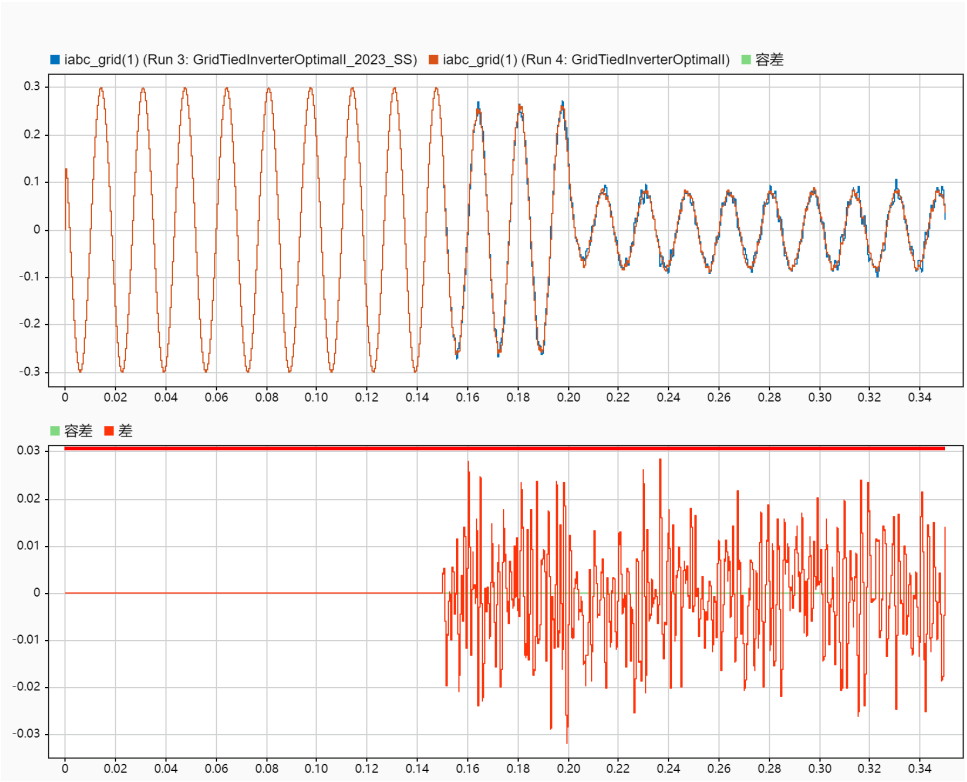

Figure C.6: Comparison of three phase grid current a (across networks).

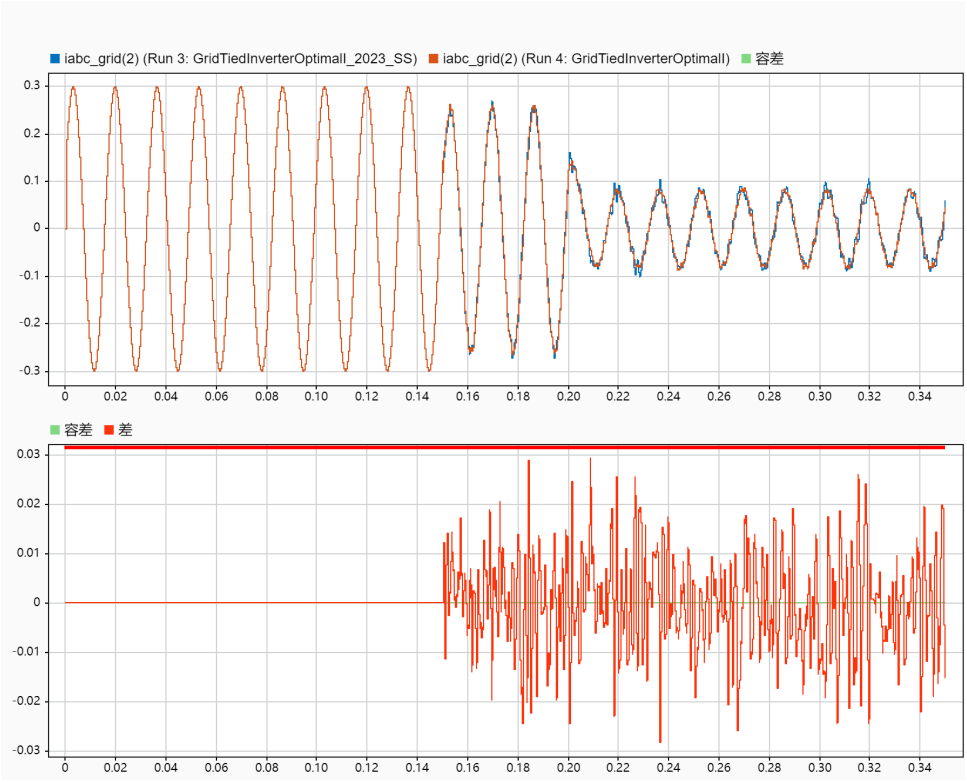

Figure C.7: Comparison of three phase grid current b (across networks).

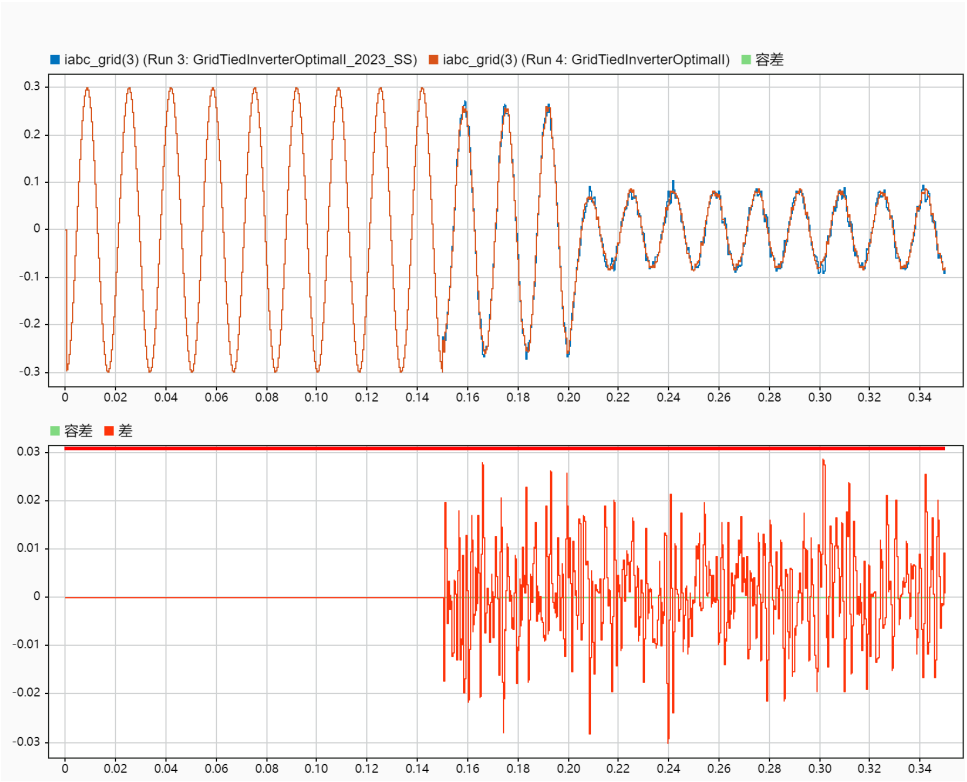

Figure C.8: Comparison of three phase grid current c (across networks).

C

D

Title of the Appendix

Appendix D: S-Function Code for Parallel Simulation Interface


```
function [sys, x0, str, ts] = Server_inside_2023(t, x, u, flag)
switch flag
    case 0
        [sys, x0, str, ts] = mdlInitializeSizes();
    case 3
        sys = mdlOutputs(t, x, u);
    case {1, 2, 4, 9}
        sys = [];
    otherwise
        error(['Unhandled flag = ', num2str(flag)]);
end
end

function [sys, x0, str, ts] = mdlInitializeSizes()
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 3;
sizes.NumInputs      = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;

```

```
sys = simsizes(sizes);
x0 = [];
str = [];
ts = [7.5e-5 0];

global sharedDataFromServer sharedDataFromClient;
sharedDataFromServer = [0; 0; 0];
sharedDataFromClient = [0; 0; 0];

disp('Server initialized successfully.');
end

function sys = mdlOutputs(t, x, u)
global sharedDataFromServer sharedDataFromClient;

sharedDataFromServer = u;
disp(['Data sent to client: ' num2str(u(:)')]);

data_recv = sharedDataFromClient;
disp(['Data received from client: ' num2str(data_recv')]);

sys = data_recv;

if any(isnan(sys))
    sys = [0; 0; 0];
end
end
```

Listing D.1: Server S-Function Code for multi-bus model

```
function [sys, x0, str, ts] = Client_inside_2023(t, x, u, flag)
switch flag
    case 0
        [sys, x0, str, ts] = mdlInitializeSizes();
    case 3
        sys = mdlOutputs(t, x, u);
    case {1, 2, 4, 9}
        sys = [];
    otherwise
```

```
        error(['Unhandled flag = ', num2str(flag)]);
end
end

function [sys, x0, str, ts] = mdlInitializeSizes()
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 3;
sizes.NumInputs      = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;

sys = simsizes(sizes);
x0 = [];
str = [];
ts = [7.5e-5 0];

global sharedDataFromServer sharedDataFromClient;
sharedDataFromServer = [0; 0; 0];
sharedDataFromClient = [0; 0; 0];

disp('Client initialized successfully.');
end

function sys = mdlOutputs(t, x, u)
global sharedDataFromServer sharedDataFromClient;

sharedDataFromClient = u;
disp(['Data sent to server: ' num2str(u(:)')]);

data_recv = sharedDataFromServer;
disp(['Data received from server: ' num2str(data_recv')]);

sys = data_recv;

if any(isnan(sys))
    sys = [0; 0; 0];
```

```
end
end
```

Listing D.2: Client S-Function Code for multi-bus model

```
clear;
clc;

global sharedDataFromServer sharedDataFromClient;
sharedDataFromServer = [0; 0; 0];
sharedDataFromClient = [0; 0; 0];

cd('C:\Users\dilu2\Desktop  \');

sub1_file = 'Sub2_2023.slx';
sub2_file = 'Sub3_2023.slx';

simTime = '4.5';
if isempty(gcp('nocreate'))
    disp('Starting parallel pool...');
    parpool('local', 2);
end

disp('Starting simulations...');
f1 = parfeval(@start_simulation, 1, sub1_file, simTime);
f2 = parfeval(@start_simulation, 1, sub2_file, simTime);

disp('Waiting for simulations to complete...');
wait([f1, f2]);

disp('Fetching simulation outputs...');
simOut1 = fetchOutputs(f1);
simOut2 = fetchOutputs(f2);

disp('Simulations completed.');

SM_speeds_Sub2_data = simOut1.get('SM_speeds2');
time_Sub2 = SM_speeds_Sub2_data(:, 1);
SM_speeds_Sub2 = SM_speeds_Sub2_data(:, 2:end);
```

```
35
36 SM_terminal_voltages_Sub3_data = simOut1.get('SM_terminal_voltages2');
37 SM_terminal_voltages_Sub3 = SM_terminal_voltages_Sub3_data(:, 2:end);
38
39 pu_Sub2_data = simOut1.get('pu2');
40 pu_Sub2 = pu_Sub2_data(:, 2:end);
41
42 Vabc_Sub3_data = simOut1.get('Vabc2');
43 Vabc_Sub3 = Vabc_Sub3_data(:, 2:end);
44
45 SM_speeds_Sub3_data = simOut2.get('SM_speeds3');
46 time_Sub3 = SM_speeds_Sub3_data(:, 1);
47 SM_speeds_Sub3 = SM_speeds_Sub3_data(:, 2:end);
48
49 SM_terminal_voltages_Sub3_data = simOut2.get('SM_terminal_voltages3');
50 SM_terminal_voltages_Sub3 = SM_terminal_voltages_Sub3_data(:, 2:end);
51
52 pu_Sub3_data = simOut2.get('pu3');
53 pu_Sub3 = pu_Sub3_data(:, 2:end);
54 Vabc_Sub3_data = simOut2.get('Vabc3');
55 Vabc_Sub3 = Vabc_Sub3_data(:, 2:end);
56
57 figure;
58 subplot(2, 2, 1);
59 plot(time_Sub2, SM_speeds_Sub2);
60 title('Sub2 - SM Speeds');
61 xlabel('Time');
62 ylabel('Speed');
63
64 subplot(2, 2, 2);
65 plot(time_Sub2, SM_terminal_voltages_Sub3);
66 title('Sub2 - SM Terminal Voltages');
67 xlabel('Time');
68 ylabel('Voltage');
69
70 subplot(2, 2, 3);
71 plot(time_Sub2, pu_Sub2);
72 title('Sub2 - pu');
```

```
xlabel('Time');
ylabel('Value');

subplot(2, 2, 4);
plot(time_Sub2, Vabc_Sub3);
title('Sub2 - Vabc');
xlabel('Time');
ylabel('Value');

figure;
subplot(2, 2, 1);
plot(time_Sub3, SM_speeds_Sub3);
title('Sub3 - SM Speeds');
xlabel('Time');
ylabel('Speed');

subplot(2, 2, 2);
plot(time_Sub3, SM_terminal_voltages_Sub3);
title('Sub3 - SM Terminal Voltages');
xlabel('Time');
ylabel('Voltage');

subplot(2, 2, 3);
plot(time_Sub3, pu_Sub3);
title('Sub3 - pu3');
xlabel('Time');
ylabel('Value');

subplot(2, 2, 4);
plot(time_Sub3, Vabc_Sub3);
title('Sub3 - Vabc3');
xlabel('Time');
ylabel('Value');


function simOut = start_simulation(file, simTime)
    cd('C:\Users\dilu2\Desktop  \');

```

```
111     load_system(file);
112
113     [~, modelName, ~] = fileparts(file);
114
115     set_param(modelName, 'StopTime', simTime);
116
117     simOut = sim(modelName, 'SimulationMode', 'normal', 'StopTime', simTime,
            'SaveOutput', 'on');
118 end
```

Listing D.3: Parallel simulation startup program

E

Title of the Appendix

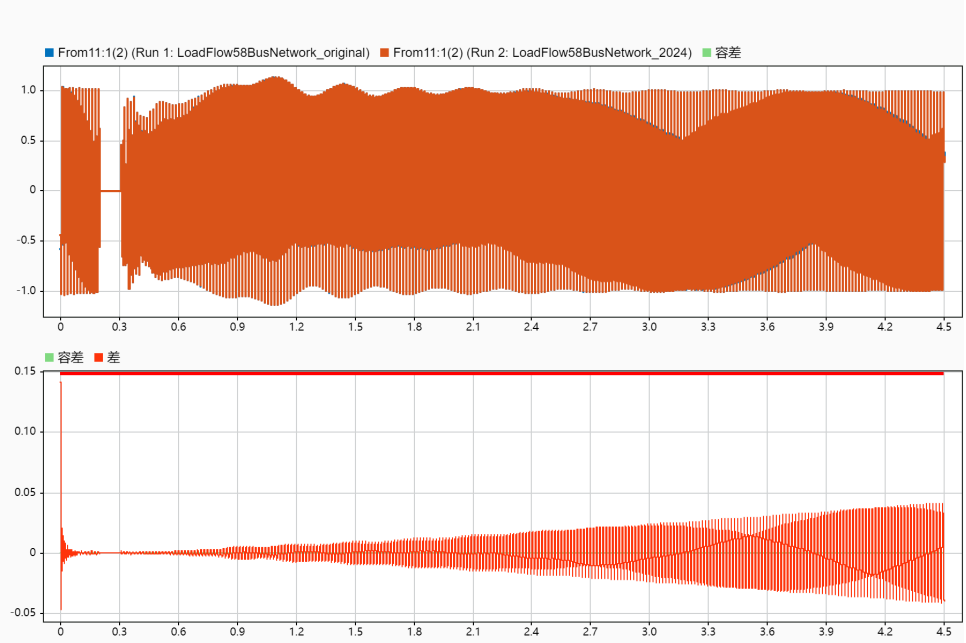

Figure E.1: Load three-phase voltage (b) error analysis.

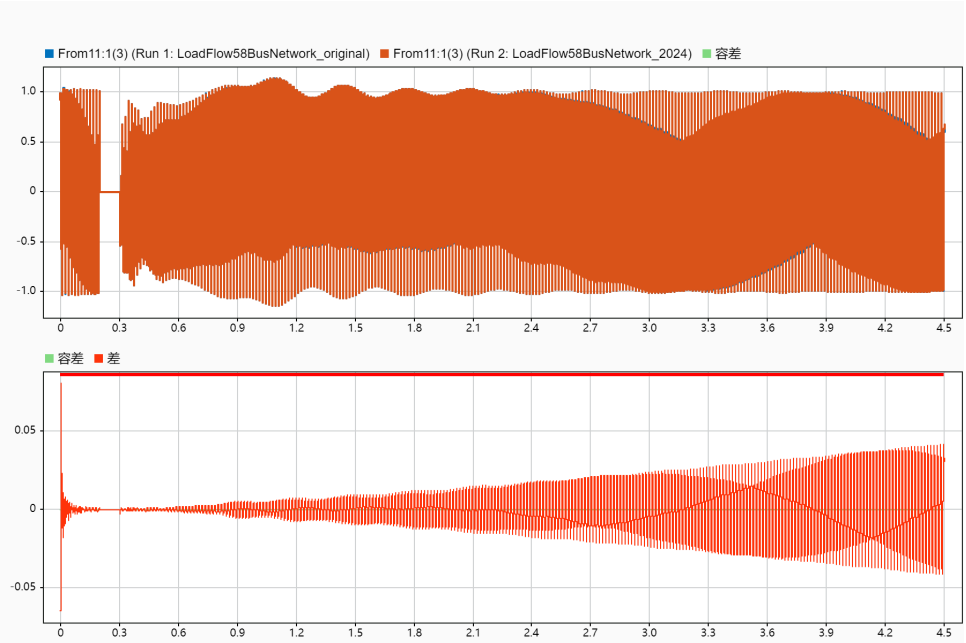

Figure E.2: Load three-phase voltage (c) error analysis.

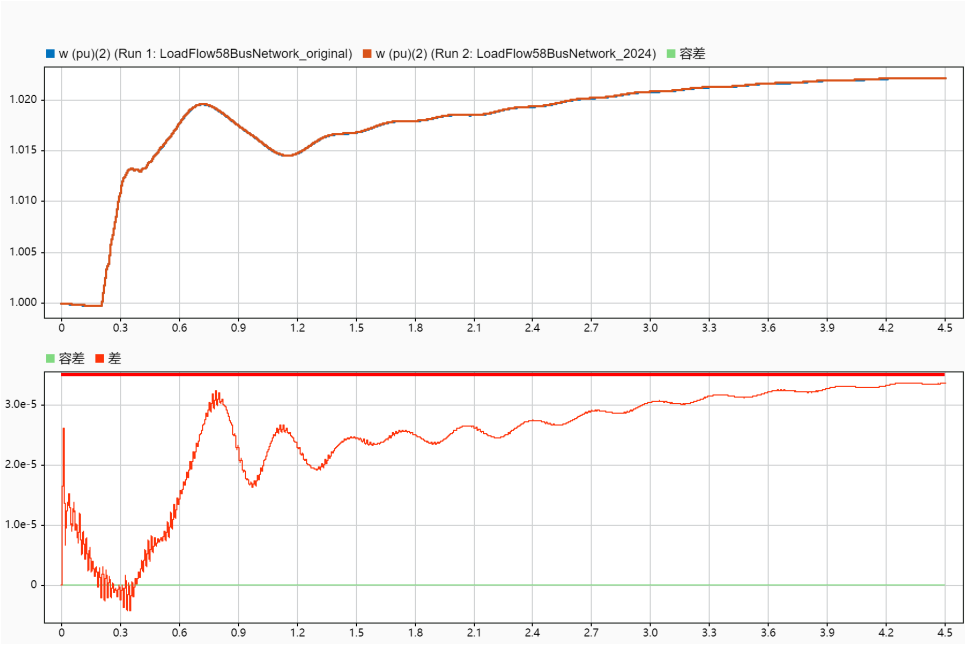

Figure E.3: Errors in motor 2 motor speed.

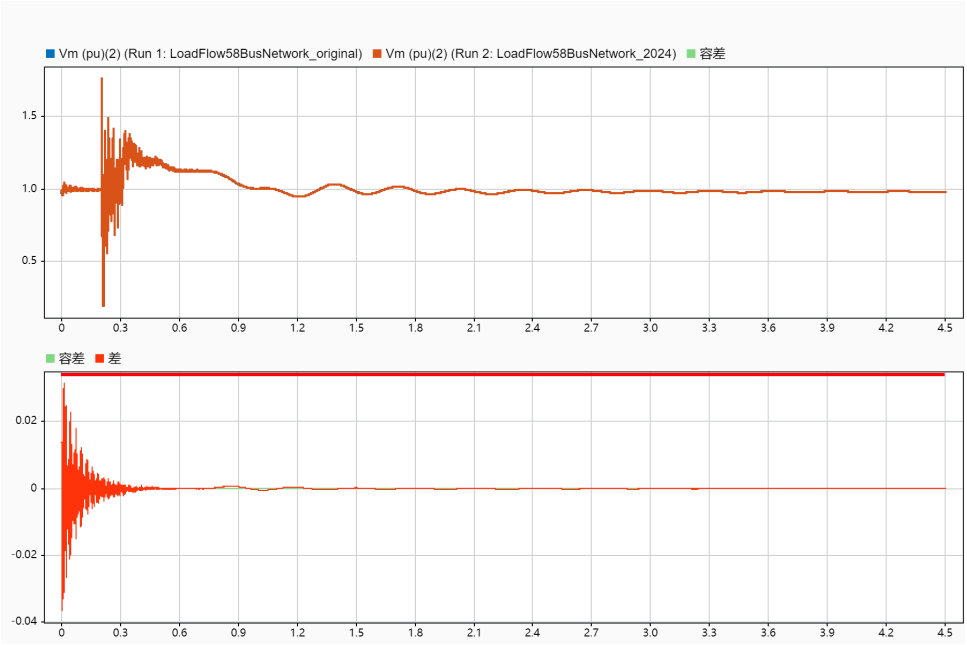

Figure E.4: Error in motor 2 synchronous motor end voltage speed

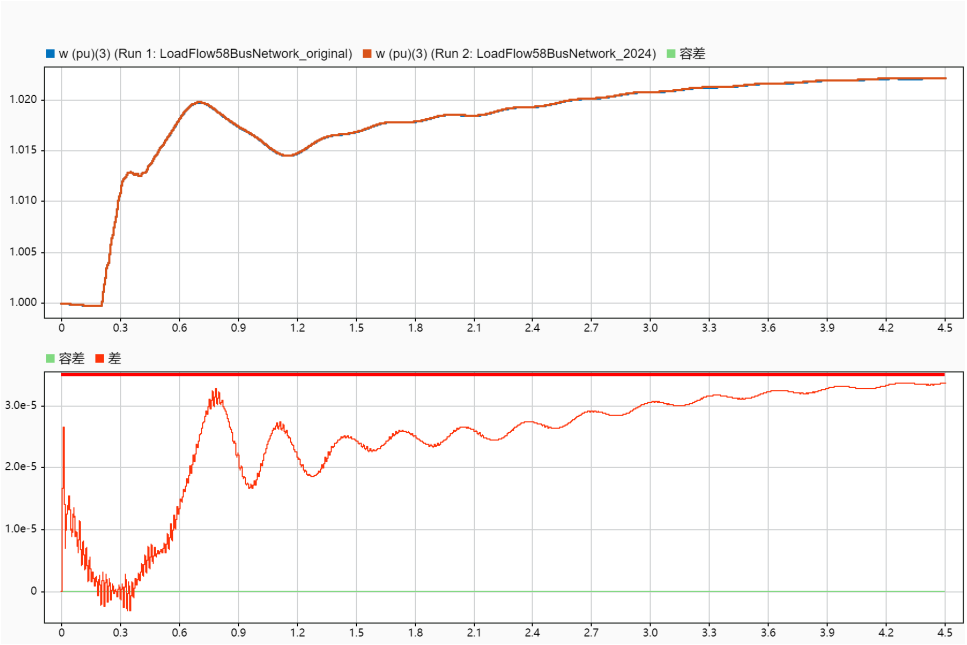

Figure E.5: Errors in motor 3 motor speed.

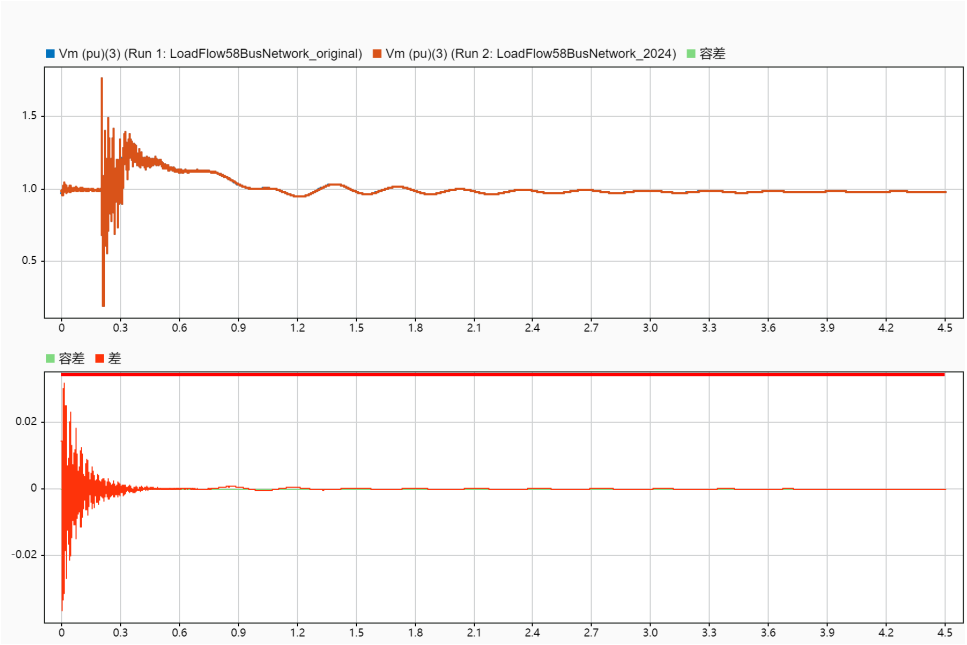

Figure E.6: Error in motor 3 synchronous motor end voltage speed

F

Title of the Appendix

Appendix A: S-Function Code for Grid-connected inverter model Co-Simulation

```
[language=Matlab,caption={Co-simulation platform using MatlabApp}]
classdef ParallelSimulationApp < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                        matlab.ui.Figure
        InitializeButton                matlab.ui.control.Button
        ModelListBox                    matlab.ui.control.ListBox
        PendingSimulationsListBox       matlab.ui.control.ListBox
        AddModelButton                  matlab.ui.control.Button
        RemoveModelButton               matlab.ui.control.Button
        StartSimulationButton           matlab.ui.control.Button
        ClearResultsButton              matlab.ui.control.Button
        TextArea                        matlab.ui.control.TextArea
        SimulationTimeEditFieldLabel    matlab.ui.control.Label
        SimulationTimeEditField         matlab.ui.control.EditField
        ResultsListBoxLabel             matlab.ui.control.Label
        ResultsListBox                  matlab.ui.control.ListBox
    end

```

F

```
21      properties (Access = private)
22          SelectedModel       % User-selected model
23          PendingModels       % List of models pending simulation
24          SimulationTime      % Simulation time
25          FutureHandles       % Handles for parallel tasks
26          SimulationResults   % Store simulation results
27      end
28
29      methods (Access = private)
30
31          % Initialize the simulation environment
32          function initializeSimulationEnvironment(app)
33              % Initialize shared data
34              global sharedDataFromServer sharedDataFromClient;
35              sharedDataFromServer = [0; 0; 0];
36              sharedDataFromClient = [0; 0; 0];
37
38              % Set the working directory
39              cd('C:\Users\dilu2\Desktop\ParallelSimulation');
40
41              % Update status
42              app.TextArea.Value = "Initialization complete.";
43
44              % Clear results list
45              app.ResultsListBox.Items = {};
46              app.PendingModels = {}; % Initialize as an empty cell array
47              app.PendingSimulationsListBox.Items = {};
48              app.FutureHandles = {}; % Initialize task handle array
49              app.SimulationResults = {}; % Initialize simulation results
50          end
51
52          % Start the simulation and obtain results
53          function startSimulation(app)
54              % Get selected models and simulation time
55              pendingModels = app.PendingModels;
56              simTime = app.SimulationTime;
57
58              % Check if at least one model is selected
```

F

```
59          if isempty(pendingModels)
60              app.TextArea.Value = 'Please select at least one model to
                    simulate.';
61              return;
62          end
63
64          % Start parallel pool if not already created
65          if isempty(gcp('nocreate'))
66              disp('Starting parallel pool...');
67              parpool('local', numel(pendingModels)); % Create parallel pool
                    equal to the number of models
68          end
69
70          % Start parallel simulations
71          app.FutureHandles = cell(1, numel(pendingModels)); % Store parfeval
                future objects
72
73          % Combine all model names for status update
74          simulatingModels = strjoin(pendingModels, ' and ');
75
76          % Update status information to show all models being simulated
77          app.TextArea.Value = sprintf('Parallel simulation %s...',
                simulatingModels);
78
79          % Start simulation for each selected model
80          for i = 1:numel(pendingModels)
81              modelName = pendingModels{i};
82              sub_file = [modelName, '.slx'];
83
84              % Start parallel simulation using parfeval
85              app.FutureHandles{i} = parfeval(@start_simulation, 1, sub_file,
                    simTime);
86          end
87
88          % Wait for all simulations to complete
89          disp('Waiting for simulations to complete...');
90          wait([app.FutureHandles{:}]);
91
```

```
92              % Update status information
93              app.TextArea.Value = 'Simulations completed.';
94              drawnow; % Force UI update
95
96              % Extract and store simulation results
97              app.SimulationResults = struct();
98              for i = 1:numel(pendingModels)
99                  modelName = pendingModels{i};
100                 simOut = fetchOutputs(app.FutureHandles{i});
101
102                 app.SimulationResults.(modelName).SM_speeds =
                        simOut.get('SM_speeds');
103                 app.SimulationResults.(modelName).SM_terminal_voltages =
                        simOut.get('SM_terminal_voltages');
104                 app.SimulationResults.(modelName).pu = simOut.get('pu');
105                 app.SimulationResults.(modelName).Vabc = simOut.get('Vabc');
106             end
107
108             % Update results list
109             app.updateResultsList();
110
111             % Close models
112             for i = 1:numel(pendingModels)
113                 close_system(pendingModels{i}, 0);
114             end
115         end
116
117         % Update the results list box
118         function updateResultsList(app)
119             % Get all result names
120             resultNames = {};
121             fields = fieldnames(app.SimulationResults);
122             for i = 1:numel(fields)
123                 modelName = fields{i};
124                 resultNames = [resultNames; ...
125                     {sprintf('%s - SM_speeds', modelName), ...
126                     sprintf('%s - SM_terminal_voltages', modelName), ...
127                     sprintf('%s - pu', modelName), ...
```

F

```
128                     sprintf('%s - Vabc', modelName)}];
129             end
130
131             % Update the list box
132             app.ResultsListBox.Items = resultNames;
133         end
134
135         % Clear simulation results
136         function clearSimulationResults(app)
137             % Clear simulation results from workspace
138             clearvars -global sharedDataFromServer sharedDataFromClient;
139             clc;
140
141             % Clear the text area
142             app.TextArea.Value = 'Results cleared.';
143
144             % Clear the results list
145             app.ResultsListBox.Items = {};
146
147             % Clear the pending simulations list
148             app.PendingModels = {};
149             app.PendingSimulationsListBox.Items = {};
150
151             % Delete MAT files
152             delete('*.mat');
153         end
154
155         % Add selected model to the pending simulations list
156         function addModel(app)
157             selectedModel = app.ModelListBox.Value;
158             if ~isempty(selectedModel) && ~ismember(selectedModel,
                    app.PendingModels)
159                 app.PendingModels{end+1} = selectedModel;
160                 app.PendingSimulationsListBox.Items = app.PendingModels; % Update
                        list box using cell array
161             end
162         end
163
```

```
164         % Remove selected model from the pending simulations list
165         function removeModel(app)
166             selectedModel = app.PendingSimulationsListBox.Value;
167             if ~isempty(selectedModel)
168                 app.PendingModels(strcmp(app.PendingModels, selectedModel)) = [];
169                 app.PendingSimulationsListBox.Items = app.PendingModels; % Update
                        list box using cell array
170             end
171         end
172
173         % Plot the selected simulation result
174         function plotSelectedResult(app)
175             selectedItem = app.ResultsListBox.Value;
176             if isempty(selectedItem)
177                 return;
178             end
179
180             % Parse the selected result name
181             parts = split(selectedItem, ' - ');
182             modelName = parts{1};
183             resultName = parts{2};
184
185             % Get simulation data
186             data = app.SimulationResults.(modelName).(resultName);
187
188             % Plot the result
189             figure;
190             time = data(:, 1);
191             signals = data(:, 2:end);
192             plot(time, signals);
193             title(selectedItem);
194             xlabel('Time');
195             ylabel(resultName);
196         end
197     end
198
199     % Callbacks that handle component events
200     methods (Access = private)
```

```
201
202         % Code that executes after component creation
203         function startupFcn(app)
204             % Initialize components on the interface
205             app.ModelListBox.Items = {'Sub1', 'Sub2', 'Sub3', 'Sub4'};
206             app.PendingModels = {}; % Initialize as an empty cell array
207             app.SimulationTime = '4.5'; % Default simulation time
208             app.TextArea.Value = "Welcome! Please initialize the simulation
                    environment.";
209         end
210
211         % Button pushed function: InitializeButton
212         function InitializeButtonPushed(app, event)
213             % Call the initialization function
214             app.initializeSimulationEnvironment();
215         end
216
217         % Value changed function: ModelListBox
218         function ModelListBoxValueChanged(app, event)
219             % Get the selected model
220             app.SelectedModel = app.ModelListBox.Value;
221         end
222
223         % Button pushed function: AddModelButton
224         function AddModelButtonPushed(app, event)
225             % Add the selected model to the pending simulations list
226             app.addModel();
227         end
228
229         % Button pushed function: RemoveModelButton
230         function RemoveModelButtonPushed(app, event)
231             % Remove the selected model from the pending simulations list
232             app.removeModel();
233         end
234
235         % Button pushed function: StartSimulationButton
236         function StartSimulationButtonPushed(app, event)
237             % Start the simulation
```

```
238             app.startSimulation();
239         end
240 
241         % Button pushed function: ClearResultsButton
242         function ClearResultsButtonPushed(app, event)
243             % Clear the simulation results
244             app.clearSimulationResults();
245         end
246 
247         % Value changed function: SimulationTimeEditField
248         function SimulationTimeEditFieldValueChanged(app, event)
249             % Update the simulation time
250             app.SimulationTime = app.SimulationTimeEditField.Value;
251 
252             % Display the updated simulation time
253             app.TextArea.Value = ['Simulation Time: ', app.SimulationTime];
254         end
255 
256         % Value changed function: ResultsListBox
257         function ResultsListBoxValueChanged(app, event)
258             % Plot the selected simulation result
259             app.plotSelectedResult();
260         end
261     end
262 
263     % Component initialization
264     methods (Access = private)
265 
266         % Create UIFigure and components
267         function createComponents(app)
268 
269             % Create UIFigure and hide until all components are created
270             app.UIFigure = uifigure('Visible', 'off');
271             app.UIFigure.Position = [100 100 720 500];
272             app.UIFigure.Name = 'Parallel Simulation App';
273 
274             % Create InitializeButton
275             app.InitializeButton = uibutton(app.UIFigure, 'push');
```

F

```
276             app.InitializeButton.ButtonPushedFcn = createCallbackFcn(app,
                    @InitializeButtonPushed, true);
277             app.InitializeButton.Position = [50 450 120 30];
278             app.InitializeButton.Text = 'Initialize';
279
280             % Create ModelListBox
281             app.ModelListBox = uilistbox(app.UIFigure);
282             app.ModelListBox.Items = {'Sub1', 'Sub2', 'Sub3', 'Sub4'};
283             app.ModelListBox.ValueChangedFcn = createCallbackFcn(app,
                    @ModelListBoxValueChanged, true);
284             app.ModelListBox.Position = [50 320 120 100];
285
286             % Create AddModelButton
287             app.AddModelButton = uibutton(app.UIFigure, 'push');
288             app.AddModelButton.ButtonPushedFcn = createCallbackFcn(app,
                    @AddModelButtonPushed, true);
289             app.AddModelButton.Position = [185 360 120 30];
290             app.AddModelButton.Text = 'Add Model';
291
292             % Create RemoveModelButton
293             app.RemoveModelButton = uibutton(app.UIFigure, 'push');
294             app.RemoveModelButton.ButtonPushedFcn = createCallbackFcn(app,
                    @RemoveModelButtonPushed, true);
295             app.RemoveModelButton.Position = [185 320 120 30];
296             app.RemoveModelButton.Text = 'Remove Model';
297
298             % Create PendingSimulationsListBox
299             app.PendingSimulationsListBox = uilistbox(app.UIFigure);
300             app.PendingSimulationsListBox.Position = [320 320 120 100];
301             app.PendingSimulationsListBox.Items = {};
302
303             % Create StartSimulationButton
304             app.StartSimulationButton = uibutton(app.UIFigure, 'push');
305             app.StartSimulationButton.ButtonPushedFcn = createCallbackFcn(app,
                    @StartSimulationButtonPushed, true);
306             app.StartSimulationButton.Position = [50 280 120 30];
307             app.StartSimulationButton.Text = 'Start Simulation';
308
```

```
309             % Create ClearResultsButton
310             app.ClearResultsButton = uibutton(app.UIFigure, 'push');
311             app.ClearResultsButton.ButtonPushedFcn = createCallbackFcn(app,
                    @ClearResultsButtonPushed, true);
312             app.ClearResultsButton.Position = [320 280 120 30];
313             app.ClearResultsButton.Text = 'Clear Results';
314
315             % Create SimulationTimeEditFieldLabel
316             app.SimulationTimeEditFieldLabel = uilabel(app.UIFigure);
317             app.SimulationTimeEditFieldLabel.HorizontalAlignment = 'right';
318             app.SimulationTimeEditFieldLabel.Position = [50 245 100 22];
319             app.SimulationTimeEditFieldLabel.Text = 'Simulation Time';
320
321             % Create SimulationTimeEditField
322             app.SimulationTimeEditField = uieditfield(app.UIFigure, 'text');
323             app.SimulationTimeEditField.ValueChangedFcn = createCallbackFcn(app,
                    @SimulationTimeEditFieldValueChanged, true);
324             app.SimulationTimeEditField.Position = [160 245 100 22];
325             app.SimulationTimeEditField.Value = '4.5';
326
327             % Create TextArea
328             app.TextArea = uitextarea(app.UIFigure);
329             app.TextArea.Position = [50 50 390 180];
330             app.TextArea.Editable = 'off'; % Disable editing to display status
                    information
331
332             % Create ResultsListBoxLabel
333             app.ResultsListBoxLabel = uilabel(app.UIFigure);
334             app.ResultsListBoxLabel.HorizontalAlignment = 'right';
335             app.ResultsListBoxLabel.Position = [520 450 120 22];
336             app.ResultsListBoxLabel.Text = 'Simulation Results';
337
338             % Create ResultsListBox
339             app.ResultsListBox = uilistbox(app.UIFigure);
340             app.ResultsListBox.ValueChangedFcn = createCallbackFcn(app,
                    @ResultsListBoxValueChanged, true);
341             app.ResultsListBox.Position = [500 50 180 380];
342
```

```
343             % Show the figure after all components are created
344             app.UIFigure.Visible = 'on';
345         end
346     end
347
348     % App creation and deletion
349     methods (Access = public)
350
351         % Construct app
352         function app = ParallelSimulationApp
353
354             % Create UIFigure and components
355             createComponents(app)
356
357             % Register the app with App Designer
358             registerApp(app, app.UIFigure)
359
360             % Execute the startup function
361             runStartupFcn(app, @startupFcn)
362         end
363
364         % Code that executes before app deletion
365         function delete(app)
366
367             % Delete UIFigure when app is deleted
368             delete(app.UIFigure)
369         end
370     end
371 end
372
373 % Simulation function
374 function simOut = start_simulation(file, simTime)
375     % Set working directory to the model directory
376     cd('C:\Users\dilu2\Desktop\ParallelSimulation');
377
378     % Load the model
379     load_system(file);
380
```

```
381     % Get the model name (excluding extension)
382     [~, modelName, ~] = fileparts(file);
383
384     % Set simulation parameters
385     set_param(modelName, 'StopTime', simTime);
386
387     % Start model simulation
388     simOut = sim(modelName, 'SimulationMode', 'normal', 'StopTime', simTime,
            'SaveOutput', 'on');
389 end
```

F

Bibliography

[1] T. Schierz, M. Arnold, and C. Clauß, "Co-simulation with communication step size control in an fmi compatible master algorithm," in *Proceedings of the 9th international Modelica conference*, Como: Linköping University Electronic Press, 2012, pp. 205–214.

[2] MathWorks, *Parallel simulation for large-scale physical models*, `https://uk.mathworks.com/videos/parallel-simulation-for-large-scale-physical-models-1689012160445.html`, Accessed: 2024-08-01, 2024.

[3] MathWorks, *Run code on parallel pools*, `https://uk.mathworks.com/help/parallel-computing/run-code-on-parallel-pools.html`, Accessed: 2024-08-01, 2024.

[4] Q. Huang, N. N. Schulz, J. Wu, T. Haupt, and A. K. Srivastava, "Power system decoupled simulation in matlab/simulink," in *2008 40th North American Power Symposium*, 2008, pp. 1–8. DOI: `10.1109/NAPS.2008.5307324`.

[5] H. Macbahi, A. Ba-Razzouk, and A. Cheriti, "Decoupled parallel simulation of power electronics systems using matlab-simulink," in *Proceedings International Conference on Parallel Computing in Electrical Engineering. PARELEC 2000*, 2000, pp. 232–236. DOI: `10.1109/PCEE.2000.873635`.

[6] E. Lelarasmee, A. E. Ruehli, and A. L. Sangiovanni-Vincentelli, "The waveform relaxation method for time-domain analysis of large scale integrated circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. CAD-1, no. 3, pp. 131–145, Jul. 1982.

[7] S. Hui and C. Christopoulos, "Numerical simulation of power circuits using transmission-line modeling," *Proc. IEE Part A*, vol. 137, no. 6, pp. 379–384, Nov. 1990.

[8] R. Braun and D. Fritzson, "Numerically robust co-simulation using transmission line modeling and the functional mock-up interface," *SIMULATION*, vol. 98, no. 11, pp. 1057–1070, 2022. DOI: `10.1177/00375497221097128`.

[9] MathWorks, *Equivalent baseband transmission line*, `https://uk.mathworks.com/help/simrf/ref/equivalentbasebandtransmissionline.html`, Accessed: 2024-08-01, 2024.

[10] N. M. Crout, K. Gregson, and M. Unsworth, "Tlm: A technique with application in the numerical solution of diffusion problems," *Agricultural and Forest Meteorology*, vol. 51, no. 1, pp. 1–20, 1990, ISSN: 0168-1923. DOI: `https://doi.org/10.1016/0168-1923(90)90038-8`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/0168192390900388`.

[11] P. Krus *et al.*, "Distributed simulation of hydromechanical systems," in *The Third Bath International Fluid Power Workshop*, Bath, England, 1990.

[12] Q. Huang, N. N. Schulz, J. Wu, T. Haupt, and A. K. Srivastava, "Power system decoupled simulation in matlab/simulink," in *2008 40th North American Power Symposium*, 2008, pp. 1–8. DOI: 10.1109/NAPS.2008.5307324.

[13] Q. Huang, J. Wu, J. L. Bastos, and N. N. Schulz, "Distributed simulation applied to shipboard power systems," in *2007 IEEE Electric Ship Technologies Symposium*, 2007, pp. 498–503. DOI: 10.1109/ESTS.2007.372132.

[14] MathWorks, *Equivalent baseband transmission line*, https://uk.mathworks.com/help/simrf/ref/equivalentbasebandtransmissionline.html, Accessed: 2024-08-01, 2024.

[15] MathWorks, *Decoupling line (three-phase)*, https://uk.mathworks.com/help/sps/powersys/ref/decouplinglinethreephase.html, Accessed: 2024-08-01, 2024.

[16] N. V. Ha, L. Van Hau, and M. Tsuru, "Dynamic ack skipping in tcp with network coding for power line communication networks," in *2020 22nd International Conference on Advanced Communication Technology (ICACT)*, 2020, pp. 29–34. DOI: 10.23919/ICACT48636.2020.9061477.

[17] M. J. A. Jude and S. K. Principal, "Multi sample retransmission timeout (rto) approximation for multi-hop wireless networks," in *2017 International conference of Electronics, Communication and Aerospace Technology (ICECA)*, vol. 1, 2017, pp. 445–448. DOI: 10.1109/ICECA.2017.8203723.

[18] MathWorks, *Run multiple simulations*, https://uk.mathworks.com/help/simulink/ug/run-multiple-simulations.html#bvnw_wk, Accessed: 2024-08-02, 2024.

[19] MathWorks, *Choosing a parallel computing solution*, https://uk.mathworks.com/help/parallel-computing/choosing-a-parallel-computing-solution.html, Accessed: 2024-08-01, 2024.

[20] H. Macbahi, A. Ba-Razzouk, and A. Cheriti, "Decoupled parallel simulation of power electronics systems using matlab-simulink," in *Proceedings International Conference on Parallel Computing in Electrical Engineering. PARELEC 2000*, 2000, pp. 232–236. DOI: 10.1109/PCEE.2000.873635.

[21] MathWorks, *Initializing a 29-bus, 7-power plant network with the load flow tool of powergui*, https://uk.mathworks.com/help/sps/ug/initializing-a-29-bus-7-power-plant-network-with-the-load-flow-tool-of-powergui.html, Accessed: 2024-08-01, 2024.

[22] H. Wu and L. Guo, "An improved security tcp handshake protocol with authentication," in *2011 International Conference on Internet Technology and Applications*, 2011, pp. 1–4. DOI: 10.1109/ITAP.2011.6006206.

[23] J. Meng and Z. Wang, "A rfid security protocol based on hash chain and three-way handshake," in *2013 International Conference on Computational and Information Sciences*, 2013, pp. 1463–1466. DOI: 10.1109/ICCIS.2013.386.

[24] G. Sybille, *Initializing a 29-bus, 7-power plant network with the load flow tool of powergui*, https://uk.mathworks.com/help/sps/ug/initializing-a-29-bus-7-power-plant-network-with-the-load-flow-tool-of-powergui.html, Accessed: 2024-08-01, Hydro-Quebec, 2024.

[25] MathWorks, *Parallel computing fundamentals*, https://uk.mathworks.com/help/parallel-computing/parallel-computing-fundamentals.html, Accessed: 2024-08-01, 2024.