THESIS

CPS SECURITY TESTBED: REQUIREMENT ANALYSIS, PROTOTYPE DESIGN AND PROTECTION FRAMEWORK

Submitted by

Md Rakibul Hasan Talukder

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Summer 2023

Master's Committee:

Advisor: Indrajit Ray

Yashwant Malaiya
Leo Vijayasarathy

ABSTRACT

CPS SECURITY TESTBED: REQUIREMENT ANALYSIS, PROTOTYPE DESIGN AND PROTECTION FRAMEWORK

Testbeds are a practical way to perform security exercises on cyber physical systems (CPS) to understand vulnerabilities and the progression/impact of cyber-attacks. However, it is challenging to replicate a large CPS, such as nuclear power plant or an electrical power grid, within the confines of a laboratory that would allow security experiments to be carried out. Thus, software-based simulations are getting increasingly popular as opposed to hardware-in-the-loop based simulations for CPS that form a critical infrastructure. Unfortunately, a software-based CPS testbed oriented towards security-centric experiments requires a careful re-examination of requirements and architectural design different from a CPS testbed for non-security related experiments. On a security-focused testbed there is a need to run real attack scripts for red-teaming/blue-teaming exercises, which are, in the strictest sense of the term, malicious in nature. Thus, there is a need to protect the testbed itself from these attack experiments that have the potential to go awry. The overall effect of an exploit on the whole system or vulnerabilities at communication channels needs to be particularly explored while building a simulator for a security-centric CPS. Besides, when multiple experiments are conducted on the same testbed, there is a need to maintain isolation among these experiments so that no experiment can accidentally or maliciously compromise others and affect the fidelity of those results. Specific security experiment-related supports are essential when designing such a testbed but integrating a software-based simulator within the testbed to provide necessary experiment support is challenging.

In this thesis, we make three contributions. First, we present the design of an ideal testbed based on a set of requirements and supports that we have identified, focusing specifically on security experiment as the primary use case. Next, following these requirements analysis, we integrate a software-based simulator (Generic Pressurized Water Reactor) into a testbed design by modifying the implementation architecture to allow the execution of attack experiments on different networking architectures and protocols. Finally, we describe a novel security architecture and framework to ensure the protection of security-related experiments on a CPS testbed.

ACKNOWLEDGEMENTS

I am sincerely grateful to Dr. Indrajit Ray, my adviser, for all of his help and advice. This thesis would not have been finished without his guidance.

I would like to thank Dr. Yashwant Malaiya to give me a chance to present my work to his Fault Tolerant course for valuable feedback. I'm grateful that Dr. Leo Vijayasarathy agreed to serve on my thesis committee.

I also like to express my gratitude to the doctorel student and co-author of one of our paper Md Al Amin for his input and valuable criticism. My another co-worker at our CyberSim project, Parker Jackim, has been very helpful taking part in brainstorming and research discussion.

My parents, my sister, my fiancee and friends helped me adjust to my new life in a foreign country, and for their unwavering support during graduate school, I want to convey my sincere thanks to them.

DEDICATION

I would like to dedicate this thesis to Almighty God.

TABLE OF CONTENTS

ABSTRACT ii
ACKNOWLEDGEMENTS iv
DEDICATION v
LIST OF TABLES viii
LIST OF FIGURES ix

Chapter 1 Introduction 1
1.1 Need for Security Experiments on CPS 1
1.2 Necessity of a Testbed for CPS 2
1.3 CPS Testbed Deisgn Challenges 4
1.4 Thesis Organization 6

Chapter 2 Background 8
2.1 Hardware-In-Loop (HIL) 8
2.2 Testbed Experiments 10
2.3 Digital Twin 10
2.4 Tuple space 11
2.5 Process Isolation 12
2.6 Access Control Model and Policy 13

Chapter 3 Related Works 14

Chapter 4 Ideal Testbed 17
4.1 General Requirements 17
4.2 Cybersecurity Centric Experiment Support in Testbed 20
4.3 Ideal Testbed Architecture 23
4.3.1 Conceptual Design based on Requirement Analysis 24
4.3.2 System Level Design 25
4.4 Barriers to Implement Ideal Testbed 26

Chapter 5 Towards a Non-ideal Yet Practical Testbed 29
5.1 Case Study: Generic Pressurized Water Reactor Software by GSE Power Systems Inc. 29
5.2 Architecture of Specialized Testbed 30
5.3 Implementation of Design 34

Chapter 6 Protection of Testbed 40
6.1 Threat Model for CPS Testbed 40
6.1.1 Outsider Threats 40
6.1.2 Insider Threats 40
6.2 Experiment Communication Model 41
6.3 Overview of Experiment Execution on Testbed 44

6.4 Inter-experiment Secure Communications 45
6.5 Testbed Tuple Space Design for Isolated Experiments 48
6.5.1 Tuple Space Manager-TSM 48
6.5.2 Tuple Space Operation 48
6.5.3 Tuple Space Transaction 49
6.6 Authorization Module (AM) 50
6.6.1 Experiment Resources and Operations 50
6.6.2 Resources and Privileges Allocation 51
6.6.3 Relationship between Control Node, Experiments, and Subjects 51
6.6.4 Object Classification 52
6.6.5 Control Node Identification 53
6.6.6 Security Policy 53

Chapter 7 Conclusion 55
7.1 Lesson Learned 55
7.2 Future Work 56

Bibliography 58

Appendix A Windows Network Configuration 63

Appendix B VMware Workstation Network Settings 67

Appendix C GNS3 Configuration 69

Appendix D Server-client communication code 73
D.1 dispatcher-server.py 73
D.2 node-client.py 76

LIST OF TABLES

4.1 Functional design based on requirements 24

LIST OF FIGURES

1.1 Number of incidents handled by U.S. ICS-CERT 2

4.1 Functional blocks of the testbed. 23
4.2 Testbed structure. 26

5.1 Replacing memory I/O with network I/O 31
5.2 Modification of component design to support ideal testbed requirements 33
5.3 Middleware services 33
5.4 Network topology in GNS3 GUI 35
5.5 Bridged network between VM and host 37

6.1 Coordination communication. 42
6.2 Collaboration communication. 43
6.3 Inter-experiment communication flow diagram. 46
6.4 Tuple space transaction. 49
6.5 Relationship between control node, experiments, and subjects. 52

A.1 Opening hardware wizard on windows: step 1 63
A.2 Opening hardware wizard on windows: step 2 63
A.3 Manual installation of hardware: step 3 64
A.4 Choose network adapter as hardware: step 4 65
A.5 Choose manufacturer and specific network adapter: step 5 65
A.6 Find added network adapter in network settings: step 6 66
A.7 Configure the adapter with specification of protocols and IP address: step 7 66

B.1 Opening GNS3 VM settings: step 2 67
B.2 Adding new hardware component: step 3 68
B.3 Setting to 'Bridged' mode: step 4 68
B.4 Selecting loopback adapter: step 5 68

C.1 Creating new GNS3 project: step 1 69
C.2 Initializing necessary virtual appliances: step 2 69
C.3 Using a cloud appliance to interface with the host network 70
C.4 Starting router and opening the console 71
C.5 Showing router interfaces 71
C.6 Showing ip address 72
C.7 Verifying the connectivity by Pinging 72

Chapter 1

Introduction

A Cyber-Physical System (CPS) consists of many individual units or systems and often is a critical infrastructure. Some example of such CPSs are power plants and distribution grids, gas transmission systems, traffic control systems, water treatment and supply systems, transportation systems, and others [1, 2]. A single security vulnerability in CPS can lead to catastrophic consequences, which ultimately can cause considerable financial and business loss, human lives, suffering, and others [3, 4]. Thus, it is paramount that CPSs are free from security vulnerabilities.

1.1 Need for Security Experiments on CPS

With the advancement of modern technology in the last few decades, critical infrastructures are also adapting new technology and tools to reduce human effort while supporting essential services more efficiently. Mechanical or hardware parts of the systems are either getting replaced or being maintained by software for automation. As digital components are introduced increasingly into the system, new security vulnerabilities occur. Nowadays, almost every sophisticated cyber-physical system consists of some major components: physical field components (motors, sensors, valves, etc.), digital controller components (PLC, micro-controller), supervisory controller and data acquisition system (SCADA), and networking for communication between components. As a result, cyber attackers now have a large attack surface to plan their attacks. Therefore, testing the system's integrity, security, and robustness requires different supporting tools.

Malicious entities can form their strategies and routes to pave the way to take control of the targeted component. For example, the attacker may want to take control of the input signal of the pump of a nuclear water reactor plant. The attacker may have installed malware in the printer used at the engineering station, which is typically far from the field components. And then, the malware can trigger some processes to propagate through the IT network, which may lead to eventually getting into the control network resulting in illegitimate access to the pump. Many techniques and tactics exist for infiltrating a large complex CPS or Industrial Control System (ICS) like [5]. The system behavior also may affect multiple parts of the system. Therefore, researchers and engineers must understand how a system reacts to possible attack scenarios so that risk can be evaluated based on experiment results. This leads us to the essential question: "How to perform security experiments for a CPS?"

1.2 Necessity of a Testbed for CPS

The strategy, consequence, prevention, mitigation, and recovery of critical CPS (CCPS) attacks differ from others. Moreover, attacks on critical infrastructure have been increasing since the Stuxnet attack occurred in 2010, targeting the SCADA system and causing substantial damage to Iran's Nuclear Program. Figure 1.1 shows how the number of incidents has been rising since 2010 [6].

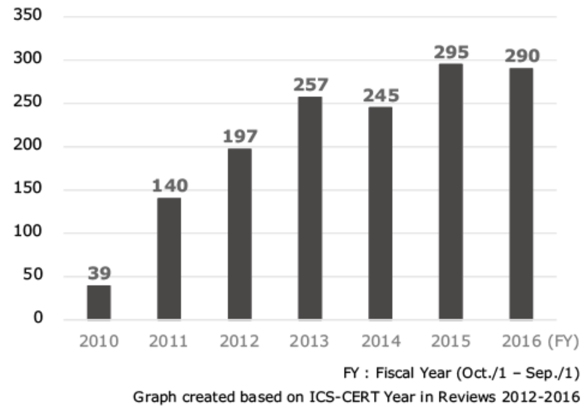


Figure 1.1: Number of incidents handled by U.S. ICS-CERT

One reason for such an increase is inadequate infrastructure or tools for evaluating the system's resilience by performing penetration testing, leading to a lack of understanding of the complex CPS system. Recently, the *MIT Technology Reviews* reported that two Dutch researchers participated in the 'Pwn2Own' championship and showed how easy it is to attack critical infrastructure. They have targeted the OPC[1] Unified Architecture (UA) communication protocol, widely used in industrial control systems.(One of the two, Keupler, had previously hacked a brand-new iPhone 12.)

The report of MIT Technology Review also brings to light an important issue,

> "There are immense differences between the consequences of hacking an iPhone and breaking into critical-infrastructure software. An iPhone can be easily updated, and a new phone is always right around the corner.On the contrary, in critical infrastructure, some systems can last for decades. Some known security flaws can't be fixed at all. Operators often can't update their technology for security fixes because taking a system offline is out of the question. It's not easy to turn a factory on and off again like a light switch—or like a laptop."[2]

Testing a functioning CPS to find security flaws or vulnerabilities is difficult. This is because testing on a live CPS cannot afford to make even a single inaccuracy or mistake; there is a real risk of the testing process harming the CPS. For example, consider a security experiment where introducing fake sensor measurements into a CPS's control network is required to mimic an attack. This experiment will introduce failure to the functioning modules resulting in inappropriate actions if it is allowed to run on a live system. Unfortunately, it is impossible to halt or pause the CPS for this reason because critical CPS provides essential services to the public. We cannot afford to interrupt services supplied by power plants, utility (gas, water, electricity) service-providing systems, traffic control systems, etc. Furthermore, security threats are constantly changing, so security testing must be done on a regular basis.

CPS testbeds are thus crucial to the CPS's security assessments [7]. The testbed environment models the interactions and real-world behaviors of the various CPS components. This makes it possible for engineers and researchers to find security flaws in developed systems before using

[1]OPC is an interoperability standard used in the industrial automation and other industries for the secure and reliable transmission of data. It is platform agnostic and facilitates the smooth flow of information across devices from various suppliers. The OPC Foundation is in charge of developing and maintaining this standard.

[2]"These hackers showed just how easy it is to target critical infrastructure" by Patrick Howell,MIT Technology review, April 21, 2022

them. It also enables them to keep their expertise up to speed with new threats. However, it is crucial to ensure that various tests conducted in the testbed environment accurately represent real-world behavior since the testbed imitates the behaviors of existent systems. This poses several challenges for the CPS testbed's design.

1.3 CPS Testbed Deisgn Challenges

The CPS testbed provides an environment to conduct experiments to study the behavior of the concerned systems. Researchers [8–11] have identified several requirements for testbeds to conduct CPS-centric experiments:

- **Fidelity**, the exactness of the testbed compared to the actual system
- **Repeatability**, the ability of producing consistent result upon executing same experiment again and again
- **Scalability**, the capability of updating the experiment setup to higher/lower number of composition elements (devices, protocols etc.) of a particular experiment without affecting the architecture of the testbed
- **Adaptability**, quality of incorporating new experiment setup with varieties of composition elements (devices, protocols etc.) without affecting the architecture of the testbed
- **Cost-effectiveness**, feasibility of the design considering the expenditure of one-time setup and long time management.
- **Measurement precision**, having support of measuring quantifiable attributes of the experiments as precise as possible
- **Safe execution of experiments**, ensuring reliability, safety and protection of the experiments, their results and the executing environment.

These requirements form the minimum set to conduct the CPS experiments effectively. However, more is needed to fully satisfy the requirements for cyber security-related experiments—more than those requirements are needed to ensure the security of the security-focused testbeds.

One of the major challenges of designing a testbed for critical CPS is replicating the actual system's physical process. Using hardware-in-loop (HIL) as part of a physical process has disadvantages. It is hard to support scalability and the ability to reconfigure the components while HIL is in place. Consequently, software simulations of the physical process are becoming more practical to fulfill the general requirements of the testbed (see details in section 4.1). Nevertheless, building a software simulator from scratch requires domain-specific knowledge. Again, it is also challenging to access a simulator representing a critical CPS. Most of these simulators are not open source and can contain proprietary or confidential information about real established critical infrastructure that is of national interest. Moreover, these simulators are not initially built for cyber-attack experiments and communication channels using ICS protocol are often not implemented. As a result, communication via networking part of the system remains unexplored. As many vulnerabilities can be hidden in the communication process, it is vital to consider and implement networking aspects into the testbed.

When a CPS testbed is solely built for cyber-security experiments and multiple organizations/collaborators/experiments working together, the protection of the testbed itself is essential. The testbed should ensure that one experiment can not get data from another without authorization (intentionally or unintentionally). Also, shared hardware resources can be an attack vector or data leakage platform.

Ensuring that one experiment process cannot go outside its run-time memory is essential for security-centric experiments. If allowed, it can cause unintended program execution or memory corruption of other infected experiments' memory and processes. This would ultimately produce erroneous results. Therefore, we must deploy individual experiments in isolated environments to confine operations and data inside the allocated working memory area. It may require that different components of a sizeable cyber-physical system be deployed as different experiments, and these experiments may need to share specific data among them. For example, HMI and the controller module are two different units that depend on each other and work together to reflect on specific goals; the devices and communication protocols used in both units can be separated and deployed as two different experiments for better analysis. Now, an attack experiment can be planned that might need to use both experiments' resources/data to complete a specific task. However, while executing the attack, it is crucial that HMI and control unit both share only the specific allowed resources/data with the authorized entity. In this scenario, ensuring safe communication to exchange information among the isolated components is necessary. It is impossible to provide a communication mechanism among isolated nodes using inter-process communication (IPC).

In this thesis work, we tried to address the following key concerns regarding a CPS testbed:

1. What requirements are ideally necessary for a cyber-security centric CPS testbed?
2. How can we design a software-based ideal CPS testbed?
3. To create a functional CPS testbed in practice, what obstacles must be addressed, and how can a viable design be developed?
4. How to implement security controls inside the testbed that can protect the testbed itself from inside threats posed by collaborative experiments?

1.4 Thesis Organization

In Chapter 2, we present the essential concepts we have used or are inspired by. These concepts are briefly explained in the scope of designing and implementing a secured CPS testbed for red-teaming/blue-teaming. Chapter 3 mentions some research works that have been well-known in the related domain, presenting a correlation with our work. In Chapter 4, first, we identify the requirements and facilities needed to design a CPS testbed. They are divided into two categories: one is for general requirements that are needed for any testbed, and the other is for testbeds that should have essential support for appropriately performing security experiments. Then we propose the architecture of an ideal testbed based on our requirement analysis. Next, we present the limitations and challenges of implementing the ideal testbed. In the next Chapter 5, we provide a case study explaining the limitations and challenges of using an industrial-level simulator that leads us to design a practically implementable testbed. This testbed overcomes the issues we have faced in the case study and provides a convenient way of performing experiments.

In Chapter 6, we address another challenge of protecting the testbed from insider threats: propose a novel communication design leveraging the tuple space model to provide inter-experiment data sharing where experiment nodes are deployed in an isolated manner. The idea of classic tuple space is based on the Linda Programming Model [12]. Our proposed framework ensures the reliability of the whole system by incorporating the mechanisms of isolation of the nodes, safe communication among experiments, and an authorization module with well-formed access control policies. Chapter 7 draws some specific conclusions of accomplishments that we have achieved, challenges we have faced and overcome. Finally we provide some specific direction for future work.

Chapter 2

Background

In this chapter, we briefly explain some of the concepts that are used in our work. Some ideas are used as an inspiration, few are used as engineering tools and some are used simply as an widely known facts.

2.1 Hardware-In-Loop (HIL)

When it comes to imitating the behaviour of actual CPS in the testbed to perform testing, it is important to have ideas about the current approaches for building a simulation environment. As physical components are essential parts of a CPS, HIL is widely used to replicate the attributes of the actual system. Before undertaking any design decision regarding choosing the simulation environment, it is important to realize what facilities HIL provides and what essential supports HIL fails to deliver:

A sort of simulation known as "hardware-in-the-loop" (HIL) simulation makes use of actual hardware in addition to computer simulation. This makes it possible for engineers to test and validate a system's performance in a setting that is more authentic than a standard simulation.Cyber-physical systems (CPSs) are frequently developed and tested using HIL simulation. CPSs are systems that combine computational and physical elements. They have many different uses, including as in industry, healthcare, and transportation.

There are several approaches to utilize HIL simulation to evaluate the performance of CPSs. It may be used, for instance, to evaluate how a CPS reacts to various inputs, such as variations in temperature or pressure. It may also be used to assess a CPS's resistance to various disturbances like interference or noise.For the creation and evaluation of CPSs, HIL simulation is a useful tool. It can aid in ensuring the security, dependability, and effectiveness of CPSs.

The following are a few advantages of employing HIL simulation for CPS testbeds:

Increased realism: HIL simulation gives engineers a more realistic setting in which to evaluate CPSs than does conventional (software only) simulation. This is due to the simulation using actual hardware resulting in more precise findings.

Early validation: HIL testing allows for the early discovery of flaws and difficulties in the development process. Potential difficulties can be found and solved before deploying the CPS in the field by testing it in a controlled setting. This decreases the likelihood of failure and improves the CPS's overall dependability.

Efficiency gain: HIL simulation can aid in making the testing process more effective. This is due to the fact that HIL simulation can test CPSs more quickly than conventional simulation.

There are some disadvantages too:

Model Complexity: Creating precise, thorough virtual models for HIL testing may be a difficult and time-consuming effort. Particularly for complicated Cyber-Physical Systems, building an accurate model of the physical system needs skill, subject knowledge, and considerable work.

Cost of Equipment: Hardware, simulators, and interfaces must be purchased in order to set up an HIL testing environment, which can be expensive. Financial difficulties might arise from the initial outlay needed to develop the infrastructure, particularly for smaller businesses or research institutions.

Testing Restrictions: HIL testing is primarily concerned with a system's hardware parts and how they interact with the simulated environment. However, it might not completely reflect the behavior at the system level, such as the interactions between hardware and software parts or the influence of outside influences. To alleviate these restrictions, additional testing techniques, such as software-in-the-loop (SIL) or system-level testing, would be needed.

2.2 Testbed Experiments

As we want to design a testbed so that we can execute security event(attacks,defense etc.) on the imitated CPS, it is important to know what a testbed experiment does, what are the goals of an experiment and how a security experiment is different from a regular one.

A testbed is an execution environment for testing a system before deployment in the real world. The testbed is made up of specific hardware and software, including an operating system, a network configuration, and the product being tested, as well as additional system and application software. Researchers and engineers use testbeds to determine the behavior of designed systems. Cyber-physical system (CPS) testbeds provide facilities to test a CPS's functionalities. The testbeds provide simulated physical model functionalities of every component included in the system [13]. But the usual testbeds do not provide cyber security testing facilities. Nowadays, every CPS contains a number of information technology (IT) devices and applications that are vulnerable to different vulnerabilities. A single vulnerability of a tiny IT component can lead to the whole organization's security system being compromised. It is important to identify and patch those vulnerabilities. security-focused testbeds are designed and developed to provide security testing mechanisms on different IT hardware and applications. The security-focused testbed contains facilities which perform different cyber security-related analyses on the components to measure security weaknesses and strengths. Such a testbed can provide or not provide the basic functionalities of the usual testbed. It is a design decision. But it certainly provides functionality to measure the security state of the cyber physical system.

2.3 Digital Twin

In our thesis, we choose to use the software-based simulation environment for appropriate reasons. First, this type of environment needs to address modeling a physical device fully in software. Though there can be arguments on how much a software model is capable of provide fidelity, it is not inconspicuous that the near future is shifting towards integrating digitally modeled components to provide a better understanding of the product or utility. Thus, a brief idea of 'digital twin' is presented here:

A software replica of a real-world system or object is called a "digital twin." It is a real-time data-driven model that aids in comprehending how the physical system or item functions as well as how it will behave going forward [14–16]. A number of sectors, including manufacturing, healthcare, and energy, employ digital twins.The following are some advantages of employing digital twins:

Better decision-making: By offering information about how a physical system or product is functioning, digital twins can aid in better decision-making. Making modifications to enhance performance and seeing potential issues before they arise are both possible with the use of this information.

Efficiency gains: Digital twins can contribute to efficiency gains by offering recommendations on how to run a system or physical object more successfully. Processes and expenses can be improved with the use of this information.

Enhanced safety: Digital twins can contribute to increased safety by offering suggestions for safer operations of the physical system or product. Utilizing this knowledge will enable one to spot possible dangers and put safety measures in place. Digital twins are a potent tool that may be utilized to enhance the functionality, effectiveness, and security of real-world systems and things. Digital twins will be much more important in a number of businesses as technology advances.

2.4 Tuple space

One of the contributions of our thesis is to design a protection framework for collaborative experiments. Here we propose deploying experiments/components in isolated spaces in terms of process and network memory. However, we also intend to maintain communication between separately deployed components via a secured data transfer mechanism. In this regard, the 'tuple space data transaction' technique is used to provide a secure and reliable data-sharing procedure.

A tuple space is a parallel/distributed computing model that uses the associative memory paradigm. It offers a collection of tuples that can be accessed concurrently. Consider two types of processors: those that create data and those that consume it. Producers place their data in the space as tuples, and consumers retrieve data from the space that matches a specific pattern. Tuple space is an associative memory, which means that tuples are accessed by their content and type rather than their address. The tuple space axiom is well-suited for context awareness. It can store context data in tuple space and make it accessible by leveraging the Linda approach's excellent decoupling properties. Nonetheless, standard matching based on exact values is insufficient, particularly for context-aware applications [17]. For this work, the Tuple Space Manager (TSM) performs data transfer operations from source to destination tuple space. Each experiment is deployed with its own tuple space, which can only be accessed by it and TSM. It is assumed that the TSM is secured and trusted and does not modify or expose any tuple space contents.

2.5 Process Isolation

Process isolation is an important paradigm that is often used to enhance security of systems.In an isolated environment, computing components can be executed where each component is separated from the others. Even the primary component (kernel) of operating system is isolated from the applications. Organizations can secure services by deploying numerous separate applications on a single system. Due to the components' concurrent execution on the same hardware, the cost is minimal. Applications on the Linux OS or UNIX platform can run in an isolated (containerized) environment, such as chrooted jails or application containers [18]. Applications are unable to communicate with or share data with other system components. Because an isolated program cannot affect other isolated apps, this technique improves security. In this article, we apply isolation techniques to ensure attack and data confinement. So that the attacks can not go beyond the experiment run-time memory, produce unwanted results, or corrupt the memory contents of other experiments. Before deploying an experiment as an isolated component in the testbed environment, the access control module executed OS commands to allocate the required resources and privileges. Without the required resources and privileges, an experiment node cannot perform the intended operations. The control node determines the required resources and privileges for the experiment node and sends them to the access control module.

2.6 Access Control Model and Policy

Access control is involved with determining the permitted activities of authenticated users and moderating any request by a user to access a system resource. Complete access is allowed in specific systems after the user's successful authentication, but most systems require more sophisticated and intricate control. A mechanism applying regulations specified by a security policy enforces the access control decision. Access control policies, in particular, can be utilized, each of which corresponds to a different set of criteria for evaluating what should and should not be allowed and, in some respects, other ideas about what security implies [19]. This work uses access control model and policies to resolve the approval decision when one experiment or entity asks for access to another entities' resources or data.

Chapter 3

Related Works

One of the common practices for building a testbed for CPS is using hardware in combination with software. These hardware-in-loop based testbeds [20–23] provide closest characteristics of the real power-grids. As they have incorporated hardware devices into the testbed, real time measurements and monitoring are supported.But one of the major challenges, discussed in [20], is having scalability and flexibility issue for setting up the larger test environment.

In [22], the scientists put out the ISAAC architecture as a means of conducting security tests on a testbed for smart grid systems. It is a distributed system that spans domains and mimics data from power generation in operations. Researchers may create, test, assess, and validate comprehensive cyber-physical security methods for the Smart Grid and other cyber-physical systems using the ISAAC platform. But as the testbed grows, incorporating sophisticated hardware, the reconfiguration and maintenance cost will rise too.

The authors in [13] describe the PowerCyber testbed developed at Iowa State University. They utilize simulation data from the Internet Scale Event and Attack Generation Environment (ISEAGE) to characterize the system architecture of a security testbed. However, there are some challenges they have faced when interoperability is required handling devices from different vendors. In addition, because of limited hardware resources, performing realistic experimental studies is difficult.

All the testbeds [13, 20, 22] mentioned above have deployed HIL to provide maximum fidelity. But while doing that, they lost their ability to scale the experiments and be flexible and adaptive with different setups of experiments. The idea of incorporating HIL into the testbed makes it harder for the testbed to grow according to the needs of the continuously evolving technology. Our work focuses on integrating software-based simulators (or any digitally developed components) into the testbed to perform collaborative security experiments.

The authors present EPIC in [11], a novel cyber-physical testbed capable of assessing cyber-attacks' effects on the cyber and physical dimensions of networked critical infrastructures (NCIs), such as power plants. The EPIC utilizes an Emulab-based testbed to simulate the cyber attributes of NCIs and multiple software simulators for the physical component. Its primary strength is its ability to support precise, meaningful, reproducible, and realistic experiments with extensive and diverse infrastructure and services. Furthermore, attack frameworks and knowledge of cyber-attack effects are critical components in selecting the best anomaly detection system (ADS). EPIC helps to find the best ADS for the system. The main crunch of this work is using a network emulator with different software-based simulator models. However, it does not provide access to the communication channel where many attacks occur. Here, integrating a simulator for physical processes depends on the targeted system's architecture, device configuration, and extensive knowledge of the respective domain, which can be a major bottleneck to gaining flexibility. Though [11] tried to build an entirely software-based testbed leveraging network simulation, there is scope for providing more flexibility and adaptive feature regarding experiment scope and simulator integration.

Choosing an effective and up-to-date network simulator can be a significant decision while using for security testing. One of the necessary requirements is being able to up-to-date virtual images of thousands of devices. This version of the Emulab in [24] virtualizes hosts, routers, and networks while maintaining near-total application openness, high system throughput, and efficient resource management. The critical design techniques are to use the least amount of virtualization that allows applications to be transparent, exploit the hierarchy found in real computer networks, perform optimistic automated resource allocation, and use feedback to adaptively allocate resources. In addition, the entire system is highly automated, making it simple to use even when scaled to thousands of virtual nodes. Finally, this paper identifies the numerous difficulties that come while working to develop a practical system and describes its motivation, design, and preliminary evaluation.

However, using Emulab has some limitations. Users can use remote clusters (servers are deployed at University of Utah) via web which may not be suitable for maintaining proper authorization and privacy of the critical system. There are ways to deploy one locally. But the process outdated and troublesome because of legacy technology. Currently, there are some other network simulators (Eg. GNS3,etc.) which have many in built feature like integrating with different modern virtualization tools (VMware,VBox etc.) or remote access tool (Eg. PuTTY) or packet analyzer (Eg. wireshark etc.) and many more.

One of the goals of our work is to protect multiple experiment configurations of different stakeholders that is a first of its kind to the best of our knowledge. To approach this problem, we have been inspired by the idea of inter-component secure communication via tuple space [25,26]. However, the previous work applied to a single-host Linux machine at the operating system level. Modern operating systems like Linux can support multiple concurrent application services on a single server instance. Furthermore, due to advancements in multi-core CPU architectures, concurrent application execution is now possible. Individual service components of such services may run in separate, isolated environments, such as chrooted jails or application containers. They may require regulated and secure access to system resources and the ability to collaborate and coordinate with one another. The authors in [25, 26] propose the Linux Policy Machine as the centralized reference monitor to provide secure inter-component communication in an isolated environment. In addition, they introduce tuple space to facilitate communication mechanisms for the isolated components. The researchers propose communication processes for multi-user operating systems but not for testbed environments. They support tuple space-based communication for isolated processes. However, we propose a communication mechanism for the testbed environment where multiple experiments composed of different configurations and devices owned by various manufacturer/stakeholder organizations are deployed as one networked system.

Chapter 4

Ideal Testbed

An ideal cyber security testbed can ensure all the requirements to be fulfilled to conduct cyber security experiments to achieve an acceptable level of research or engineering analysis of the system's architecture to assist in taking important security measures. Once we identify the characteristics of an ideal testbed, we can design the testbed. It is known that all the ideally assumed or theoretically designed features may not be applicable practically or in the research domain. However, having a benchmark of required support for building a cyber-security experiment-oriented CPS testbed is essential.

4.1 General Requirements

Requirements are usually identified to fulfill the objectives of the functionality or feature. A testbed's main functionality is to provide a platform to perform testing as accurately as possible, which demands many general requirements to be considered. When the testbed has to perform a special kind of testing: cyber security related experiments, some additional requirements become significant too. This section will explain the standard requirements for any CPS testbed. We also present the challenges and limitations of implementing those in practical scenarios.

Fidelity This attribute measures to which degree the testbed can represent the actual system events. A critical CPS testbed should be capable of modeling all the components, actions, reactions, hardware, software, and communication layers typically used in real infrastructures as closely as possible. Depending on the methodology of implementation: hardware-based, hardware-in-loop (HIL) based, or fully software-based, fidelity can vary. If the testbed is implemented by all the actual hardware used in the physical process (sensors, actuators, etc.); software used for digital process (HMI, SCADA, information management, etc.); protocols practiced in Industrial Control Systems for communication (Modbus, DNP3, etc.); it will present the highest level of fidelity. One of the examples of such a testbed is deployed by Idaho National laboratory [27]. However, such a testbed's limitation is its high deployment and maintenance cost. Software-based testbeds can be cost-effective in this regard and can also become highly configurable, which is another limitation of hardware-based testbeds.

Repeatability A testbed's ability to replicate the same set of experiment sequences with the same set of parameters and provide consistent results is important [28]. In order to accomplish this, researchers should be able to modify the experiment's beginning settings, control the sequencing of the experiment, and add essential events at the appropriate timestamps [29]. For example, in a perfect CPS security testbed for critical infrastructure, the likelihood of a threat to the system must be assessed through repeated attack-defense events.

Scalability Any system that can grow larger regarding set-ups, users, or features should always be tested for high scalability. For example, a critical CPS or industrial control system can have hundreds of field devices. Moreover, depending on the domain, network traffic can also be high to achieve desired productivity in the service. The ideal testbed should have the capability of scaling up the experiment set-up without changing the architecture or fundamental components of the testbed. For example, a testbed can perform experiments on a traffic control system for a small city with fewer field devices and traffic. It is an essential requirement that the testbed can also perform experiments in the scope of a more extensive city traffic control system.

Flexibility A CPS testbed should be capable of performing various kinds of experiments and analyses conveniently. For example, A security experiment-oriented testbed for nuclear power plant should be able to conduct different kinds of attack experiments to find the system's vulnerabilities. It should also support testing the plant's resiliency by incorporating the plant's defense mechanism while attacks continue. The testbed should be able to change the experiment strategies and evaluate its impact when necessary.

Adaptability Adaptability is a crucial aspect of any cyber physical system testbed. Such testbeds must be designed to accommodate changes and updates in both hardware and software components, as well as to handle variations in operating conditions and user requirements. In order to achieve this adaptability, testbeds need to incorporate flexible architecture, modular design, and open interfaces that allow for seamless integration of new components and systems. Additionally, the use of standardized protocols and interfaces ensures interoperability and compatibility with other systems. With these features in place, a cyber physical system testbed can quickly and effectively adapt to changing needs and requirements, allowing for more efficient and effective testing and development of new technologies [30].

Cost-effectiveness A testbed is not a full-fledged real system. The purpose of the testbed is to perform necessary experiments on relatively very low-budgeted setups than actual ones. One of the important aspects of being cost-effective is running the emulation or simulation of the experiments repeatedly without further expense. [31].

Measurement Accuracy The process of performing experiments in the testbed should be quantifiable. The testbed should allow extracting measurements of the defined attributes in a feasible period of time. The methodology of executing experiments on the testbed should not affect the measurements of the targeted attributes.

Safe Execution Experiment In general, any CPS testbed should be able to perform experiments without disrupting the environment of the testbed itself. Moreover, experiment results should be error-free and maintain integrity. In the case of the cyber-security perspective, this requirement is even more significant. A cyber security testbed has to serve two major purposes: 1) The testbed cannot allow the ongoing experiments to infiltrate the testbed's functionality or component. 2) The testbed should allow security experiments to execute attacks, exploit the vulnerabilities and measure the consequences only on experiment setups or pieces of equipment in a controlled manner.

Researchers must ensure that their experiments are conducted in a controlled and secure environment to prevent unauthorized access or data breaches. This may involve isolating the testbed from external networks, using robust authentication mechanisms to limit access to authorized personnel only, and encrypting sensitive data to prevent interception or tampering. Additionally, researchers should follow established ethical guidelines and obtain necessary permissions from relevant stakeholders before conducting experiments involving potentially sensitive information or activities. Monitoring the testbed regularly and promptly responding to potential security breaches or incidents is also essential. By prioritizing security in addition to safety, researchers can conduct experiments on a testbed for security with the confidence that they are protecting both the experiment and any relevant stakeholders from potential harm.

Diversity Like CPS devices/products(PLCs, RTUs, sensors, valves, motors, etc.), vulnerabilities are also different from one product to another. Moreover, ICS protocols also vary from one domain to another. Therefore, the ideal testbed should be able to incorporate any combination of different types of products, processes, and protocols to provide a wide range of experiment criteria

4.2 Cybersecurity Centric Experiment Support in Testbed

There are some essential requirements for the testbed which play vital roles in providing a secure, efficient, and effective simulation environment and preventing improper experiment results. The most common requirements, identified by researchers [8–11], are *fidelity, repeatability, scalability, flexibility, cost-effectiveness, measurement accuracy, diversity, and safe execution of experiments*. In the following, we identify some essential cyber security-focused testbed requirements in addition to the above-mentioned requirements. We also discuss their impact and influence on the testbed environment for conducting cyber-security experiments in a safe manner.

Experiment virtualization: Critical cyber-physical systems like power plants work on many computing machines that communicate with each other over a networking protocol. Hence, a testbed must support this higher number of processing machines and a communication layer. However, managing such a large number of physical resources for testing is more expensive. Consequently, virtualization of a large number of computing machines out of a limited number of physical resources is essential to ensure effective usage of physical resources [24]. Moreover, the testbed requires flexibility in designing, deploying, and executing experiments. Testbeds solely established on physical resources pose difficulties in managing and configuring the physical entities required by the experiments. However, experiments can demand a variable number of resource allocations at different times of the project's lifetime. Virtualization of nodes can ease this resource update with low manual activity.

Nodes and experiments isolation: To prevent unintended or malicious data exchange across experiments, an isolation mechanism is required to protect the data and process of an experiment. Assuming that each experiment has a single owner, this grants control over the nodes on which the experiment is executed. No inter-experiment network communication is allowed unless an authorized mechanism supports one. Moreover, parallel execution of processes (component functionality) on a single node may increase the risk of unintended influence. However, initializing an isolated container or VM per component can provide an extra layer of control and separation from the other components' functionality.

Secured inter-experiment communication: Each experiment deployed on the testbed may represent a single component of an extensive system. Because separate units rely on one another to represent the activities of the entire system, it is necessary to ensure secure communication between the experiments. But before one experiment collaborates with another, a coordinated approval is necessary to make the collaboration secure. Moreover, the testbed must ensure the communication messages' confidentiality, integrity, and availability. Violation of these three aspects of security can compromise sensitive information or produce incorrect results. Since experiment outputs influence the real critical CPS design, it would be vulnerable when erroneous results are considered to design and deploy the real system.

Attack libraries: Users need to launch multiple attacks on different experiments based on the deployed system components in the testbed. Support of built-in attack libraries provides accessible interfaces to perform attack experiments. Distribution and execution of attack libraries can be done in three ways:*(i) scripts provided and run by the testbed*, *(ii) scripts pulled from third-party organizations*, and *(iii) scripts developed by the owner*. The best practice is that users don't need to write scripts or access third-party sources. Before adding attack scripts to the testbed, they must be tested and evaluated correctly to ensure their intended outputs. Also, the testbed must maintain attack script integrity once they are included in the testbed.

Monitoring module and attack analytic: An experiment owner can spawn a dedicated monitoring node with predefined objectives. The monitoring module performs tasks to collect and analyze experiment activities. An attack analytic, a part of the monitoring module, generates insights for further actions with visualization and relevant reports from the collected data.

Experiment checkpoints: Sometimes it is necessary to roll back experiments to a certain executed state so that users can review experimental decisions and reconfigure the setup. Experiment checkpoints are necessary to recover from wrong states caused by cybersecurity experiments [32]. They also provide a quick recovery option so that experiments become fault-tolerant.

Attack confinement: Various security analyses are carried out across multiple experiments based on the deployed system components at different nodes. But security attack experiments may attempt to expose the component's security flaws. Attack scripts are executed with predetermined actions to observe their consequences on a predefined perimeter of the system. It is essential to protect the testbed to confine an attack and its effect to prevent intentional or unintentional damage to another experiment. So an attack can not go from one experiment run-time environment to another without proper authorization.

Experiment data confinement: Each experiment node has a different set of data, like configuration and design files, experiment results, attack scripts, etc., containing sensitive information about the system organization. Any data leakage event may compromise the sensitive information, which can cause many unwanted consequences. Moreover, data integrity is also essential to ensure that experiments generate correct results, which are considered while making design decisions in a real critical CPS system to protect it from known and unknown security threats. However, data integrity may also be tampered with while data is shared and processed across experiments. Hence, the testbed needs to avoid these data leakage events where isolation can provide data confinement services.

4.3 Ideal Testbed Architecture

After figuring out the requirements for a cyber-security centric CPS testbed, we have identified four implementation concepts which will provide the best scope to maximize the potential of ideal requirements. The following figure illustrates the idea of combining four concepts: Simulated model of physical process, virtualization of resources, network simulation, and integration of security experiment appliances.

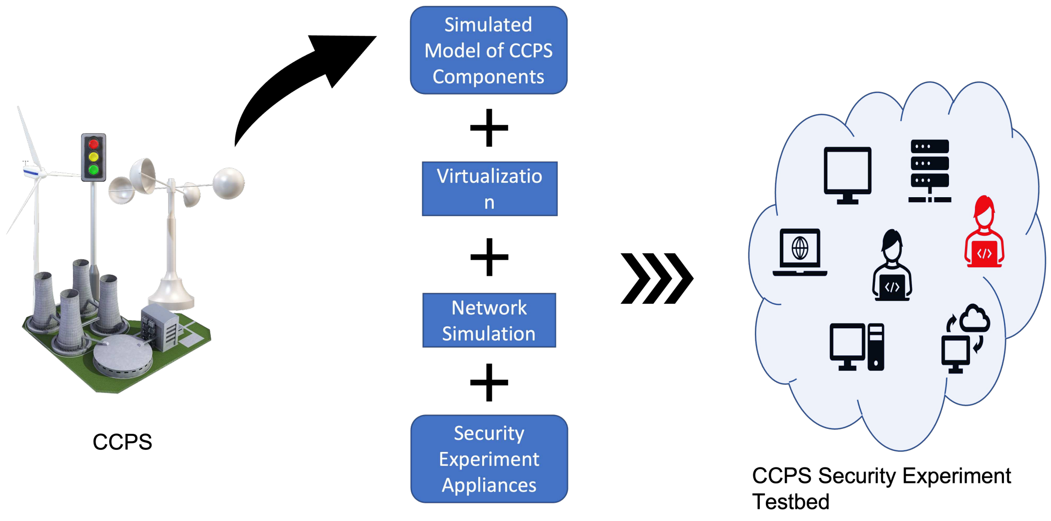

Simulated Model of CCPS Components
+
Virtualizatio n
+
Network Simulation
+
Security Experiment Appliances
CCPS
CCPS Security Experiment Testbed

Figure 4.1: Functional blocks of the testbed.

4.3.1 Conceptual Design based on Requirement Analysis

Designing a system requires maintaining a trade-off between the requirements and necessary features. Requirements are prioritized according to the primary goals of the system. Here, we are building a testbed to perform cyber attack experiments on different types and scaled Cyber-physical systems. We want to make an assumption that most of the cyber attacks are initiated and propagated by exploiting vulnerabilities in the cyber/software/network layer. The consequences can be visible in the physical layer. Following this assumption, we prioritize our requirements and necessary support for the testbed in the Table 4.1:

Table 4.1: Functional design based on requirements.

Implementation Concepts	**Requirements & Supports**
Simulated model of physical process	Cost-effectiveness Scalability Flexibility
Virtualization of assets resources	Cost-effectiveness Isolation of experiments Repeatability Experiment Virtualization Experiment Snapshot Checkpoints
Network simulator and traffic data monitoring	Diversity Fidelity Measurement Accuracy Isolation of experiments
Security tools and framework	Safe Execution Attack Confinement Experiment Data Confinement Attack Libraries

As our first and foremost goal is to provide as much flexibility and scope for penetrating the vulnerabilities in the cyber layer, we have focused much on the diversity, reconfiguration capability, and safe execution of the experiments. As long as we maintain the physical property as close as to the actual system based on software-based simulation, there is a possibility of compromising the physical process's fidelity. Being cost-effective is one of the significant goals while imitating a

CPS's characteristics because the number of hardware used in a CPS will cost a lot, even for building a testbed. However, using expensive hardware is an obstacle to achieving the high scalability of the targeted system. Thus, software-based solutions can be very effective for the simulation and virtualization of assets. Using a network simulator as a communication platform can effectively gain fidelity and diversity for the actual attack surface in the cyber layer. Moreover, measuring data can be more accessible via tracking software processes or network traffic data. Therefore, one crucial goal for a cyber security-centric testbed is providing the necessary tools for attack experiments and maintaining the three aspects of the experiment data: confidentiality, integrity, and availability. A secure inter-experiment communication framework is also necessary to protect the multiple experiments collaborating with each from going awry [33].

4.3.2 System Level Design

As most critical CPSs are distributed and networked, a testbed should imitate the functionalities of a distributed networked system. Otherwise, the threats related to any distributed networked system can not be explored while performing security experiments in the testbed. This is why, in order to provide a platform for message-passing capabilities among the nodes, a network event simulator is required. This simulator is also responsible for managing and distributing resources among the experiment nodes. A bunch of experimental nodes together build the whole structure of a cyber-physical system. Figure 4.2 illustrates different components of a testbed that we require. The major components are *(i) testbed controller*, *(ii) experiment server*, *(iii) virtualization of experiment nodes*, *(iv) experiment controller layer*, and *(v) experiment nodes*.

The testbed controller (i) is responsible for managing resources for experiment nodes (v). It also provides services to allocate network address spaces for the experiment nodes as required by the experiment requirements. The controller has overall knowledge of the availability of resources, user space, node visualization, shell management, etc. It must be installed on a machine other than the experiment servers, allowing us to use resources in a request-response manner. This separation

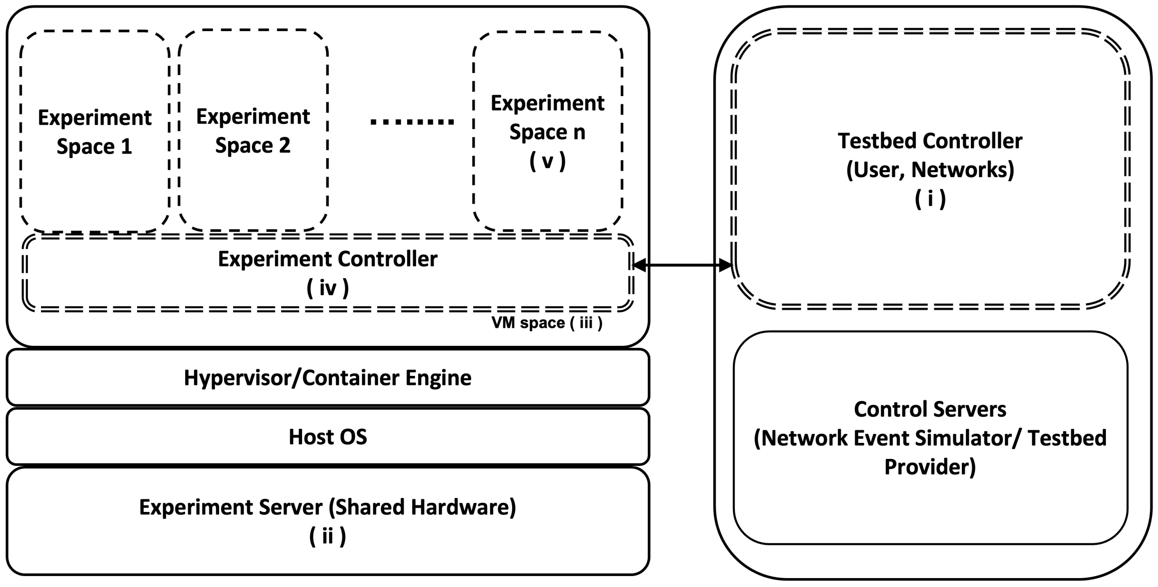

Experiment Space 1
Experiment Space 2
....
Experiment Space n (v)
Experiment Controller (iv)
VM space (iii)
Hypervisor/Container Engine
Host OS
Experiment Server (Shared Hardware) (ii)
Testbed Controller (User, Networks) (i)
Control Servers (Network Event Simulator/ Testbed Provider)

Figure 4.2: Testbed structure.

prevents experiment nodes from manipulating the control server's memory space. The system on which this controller is installed is known as the testbed control server.

Experiment servers(ii) provide the computing platform for experiment nodes. A single or multiple machines combined can provide the processing capability for experiment nodes. The number of experiments and connectivity among them is generally greater than the actual physical resources. Therefore, an abstraction of virtual machines (iii) and networking topology are supported based on containerization or hypervisor. One control node mapped to each owner of the experiment(s) will reside in the experiment control layer(iv). It is responsible for maintaining a gateway to the experiment space containing multiple nodes. A user, the experiment owner, can instantiate multiple nodes for the experiment. An experiment cannot have more than one owner. Experiment nodes are allocated resources from the experiment server(s). A hypervisor will run on the hardware of the experiment server to provide virtualization to the nodes.

4.4 Barriers to Implement Ideal Testbed

Researchers have minimal access to resources related to infrastructure that are of national interest and critical in nature. Moreover, they face many challenges to design a practical testbed lacking information about real critical CPS. Here are some significant challenges we have faced while trying to implement the design for an ideal testbed:

Unavailability of Digital Components So far, we have designed our testbed considering some ideal situations. We have assumed there will be simulated models or digital twins [14–16] available for each component. However, the technology of building an exact digital version of real CPS components has yet to develop enough. However, there has been much research recently on how to make digital/virtual/simulated versions of sophisticated components of a complex system. This future is not so far where the real challenge will be performing security experiments across various types of digital components, maintaining confidentiality, privacy, and integrity. One major work of this thesis is designing a security framework for a testbed that supports multiple experiments (can be of different organizations) sharing data with each other. Chapter 6 presents the work on protection for the testbed.

Non-modular Simulators Many simulators for critical CPS are not modular in most cases. It is not easy to separate each component and deploy it individually. Usually, the software represents the whole system like a monolithic program. As a result, it takes work to determine the exact timestamps of executing a significant step (getting or setting values of state variables) in a running simulation. Again, it is also difficult to trace the effect of experiments across different components.

Proprietary Software/Simulator Most of the simulators for critical systems (like nuclear power plants) are proprietary resources that contain sensitive information about special technology which can be of national interest. Consequently, access to a simulator of this kind is controlled rigorously. So, finding an open-source simulator where researchers can work on it is not easy. Usually, these kinds of critical CPS simulators are part of the technology control plan. Therefore, individuals must undergo special training to access and use it for research.

Inadequate Access to Simulator As these simulators are not designed keeping security testing in mind, there is little scope for accessing different parts of the software. However, if researchers want to design and perform attacks, the system should have the common vulnerabilities an attacker might look for. As a result, it is essential to have enough scope for accessing and tracking different components of the simulator, communication channel, APIs, libraries, and many more.

Integration Difficulties One of the crucial requirements of the testbed is diversity and adaptability. For example, replacing a good PLC with a rogue one or incorporating digital versions of various vendors' appliances are common cases for security testing. Most simulators implement only a specific set of hardware or physical process. Modifying the nature of already implemented digital components in the simulator is challenging. For example, including a vulnerability via integrating a customized composition needs a flexible interface for adaptation.

Undermining of Communication Channel Software-based simulators can achieve the necessary requirements without being too expensive. Nevertheless, most software-based simulators are developed using MATLAB/SIMULINK based on mathematical models or equations. Though these simulators can match the environment of the physical process of the actual critical CPS very well, the cyber or networking part of the system is not yet well considered. As a result, many attributes of networks typically used in critical CPS remain absent from the testbed scenario. However, many cyber-attacks are initiated or propagated using vulnerabilities in the networking attributes (topology, protocols, access controls, etc.).

Chapter 5

Towards a Non-ideal Yet Practical Testbed

To construct a practical testbed that overcomes the challenges of an ideal testbed (described in the preceding chapter), we first investigate available simulator software that can model the physical process of a CPS. We intend to create a usable testbed by combining industry-level simulator tools. We first analyze the simulator software's constraints and then create implementation approaches to alleviate those restrictions while maximizing the functionality that an ideal testbed should have.

5.1 Case Study: Generic Pressurized Water Reactor Software by GSE Power Systems Inc.

We have been able to get access to a simulator for our research. This fully software based simulator basically presents a generic pressurized water reactor. It has been developed by GSE Power Systems, Inc[3]. This simulator can simulate the physical process of a water reactor. In order to incorporate this simulator into our designed testbed, we had to modify our ideal implementation design to support necessary requirements and facilities.

Absence of networking The simulator is a monolithic program. Components of the simulator do not communicate with each other via any networking protocol. The software is based on memory I/O using inter process communication. As there is no networking between the components, it is not possible to explore the vulnerabilities related to networking using the simulator. As the software does not give any abstraction of network I/O operations, general Industrial Control System (ICS) networking protocols are also absent here. As a result, this simulator needs to be integrated with a network simulator. This will allow the testbed to perform security experiments exploiting network vulnerabilities.

[3]https://www.gses.com/engineering/systems-and-simulation

Proprietary Software This is not an open source software but a proprietary solution. Only Restricted users have limited access to software via APIs. There are two backdoor APIs – GET and SET to communicate with the simulator. These APIs can only get or set the values for specific variables of significant components.

5.2 Architecture of Specialized Testbed

As we have specified the limitations of the simulator in the previous sections, we target to modify the design of our ideal testbed to overcome the shortcomings of the software simulator. To address the issue of absence of network communication module, we target to replace the memory I/O with network I/O by creating a parallel world for the simulator using a network simulator like GNS3.[4]

Network simulator is a software which helps to design, build and test networks of different scales, typically without using any hardware. GNS3 is chosen as network simulator tool because-

- GNS3 is an open source, free software.
- No real hardware router/switch/PC are needed.
- Different kind of network topology can be created using different ISO image of router, OS image for PC etc.
- GNS3 network can be linked to a real network.
- Different tools for packet monitoring (E.g., Wireshark) are supported.

We create a parallel world for the water reactor simulator inside the GNS3, where there will be a node for each component of the simulator. Suppose, depending on the water level a valve door is controlled in the water tank. For simplicity we consider only three components involved in the example scenario: Controller (C) , water level sensor (S) and valve door actuator or motor (A). The

[4]https://www.gns3.com

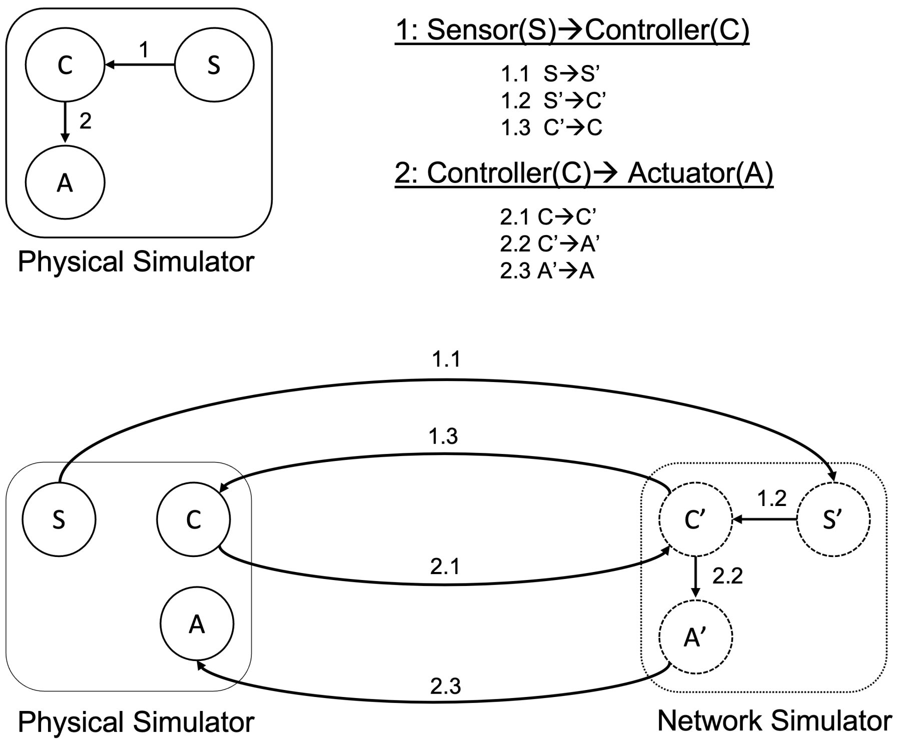


Figure 5.1: Replacing memory I/O with network I/O

whole scenario can be divided into two communication. First, the controller(C) receives data from the sensor(S), after computing the necessary actions the controller sends the action result to the actuator(A). These two communications (1 & 2 in figure 5.1) need to be replaced with networking abstraction.

We target to create a node for each component in the GNS3 network simulator and define a networking path between those according to defined communication paths in the physical simulator software. According to our example, we create C' for C, S' for S and A' for A in the GNS3 network simulator. Then we define the following networking paths

$$S' \rightarrow C' and C' \rightarrow A'$$

which are similar to the ones in the physical simulator software. Now instead of passing data from S to C directly, the communication 1 will take the following route:

$$1.1 : S \rightarrow S'$$

$$1.2 : S' \rightarrow C'$$

$$1.3 : C' \rightarrow C$$

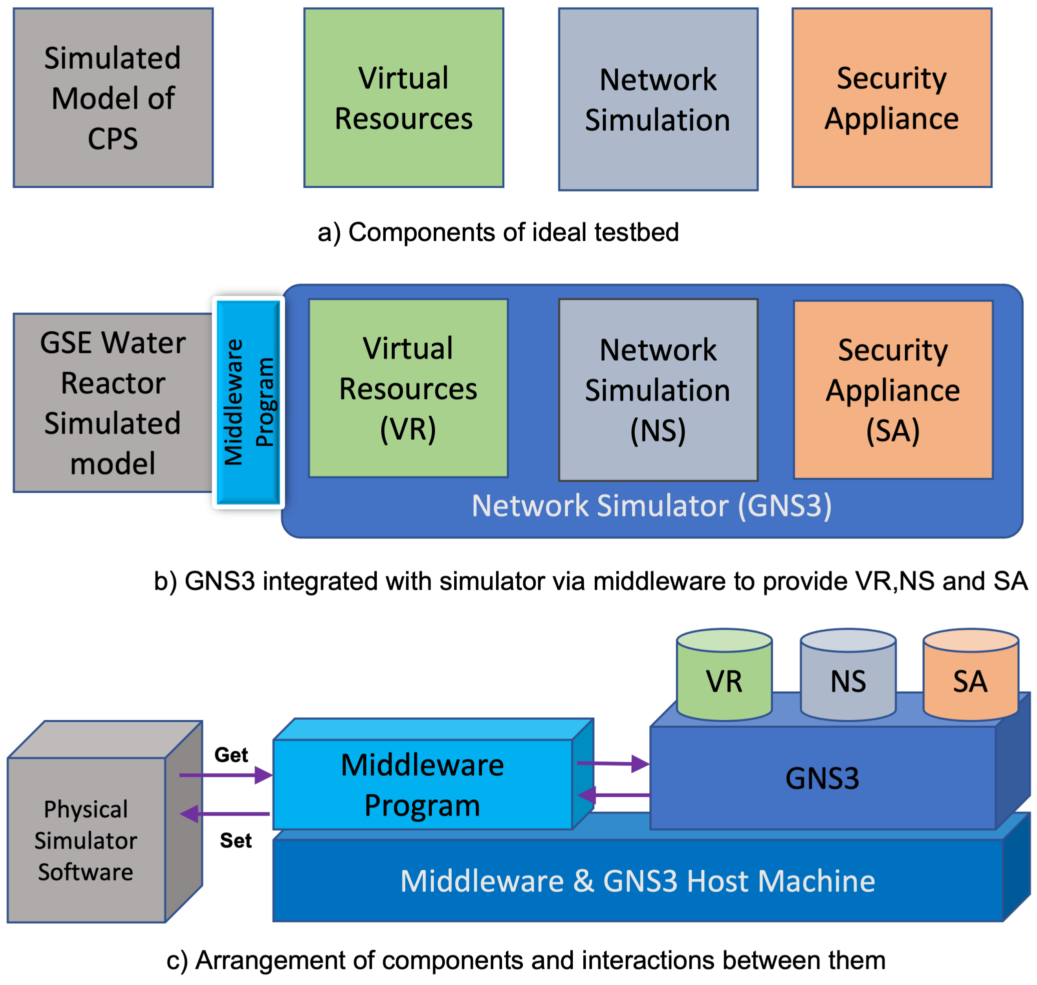
We can see in the figure 5.1 the step 1.2 is the replacement of communication no 1 where 1.2 represents the networking communication between two components. Step 1.1 and 1.3 are interfacing communications between the physical simulator software and the network simulator nodes. Each node in GNS3 has only two functionalities: receive data from source and send data to destination. The backdoor APIs (GET &SET) are used to transfer data between reactor simulator entity and network simulator node (Step 1 & step 3). This data transfer is handled via a middleware component. To allow the necessary components and features of our previous design of ideal testbed, our modified testbed design looks like in figure 5.2

Here GNS3 and middleware both are deployed on a host machine. The physical simulator software resides in different machine/server. The middleware component provides necessary services to maintain communication with the physical simulator software and the GNS3. Virtual resources, network simulation between them and other necessary security appliances and framework are deployed on GNS3. The middleware is responsible for three services(figure 5.3):

API agent service: This service makes the calls to APIs (SET & GET) to pull or push data from or to the physical simulator software. Moreover, It also collaborates with configuration service to create a queue of instructions so that the dispatcher server can execute them accordingly.

Communication Service: A dispatcher server and one client for each node in GNS3 together build this service. This service provides communication channel with client nodes. The dispatcher server takes care of three things:

1. It passes data to appropriate client node according to instructions stored in an instruction buffer.



Simulated Model of CPS
Virtual Resources
Network Simulation
Security Appliance
a) Components of ideal testbed
GSE Water Reactor Simulated model
Middleware Program
Virtual Resources (VR)
Network Simulation (NS)
Security Appliance (SA)
Network Simulator (GNS3)
b) GNS3 integrated with simulator via middleware to provide VR,NS and SA
VR
NS
SA
Physical Simulator Software
Get
Set
Middleware Program
GNS3
Middleware & GNS3 Host Machine
c) Arrangement of components and interactions between them

Figure 5.2: Modification of component design to support ideal testbed requirements

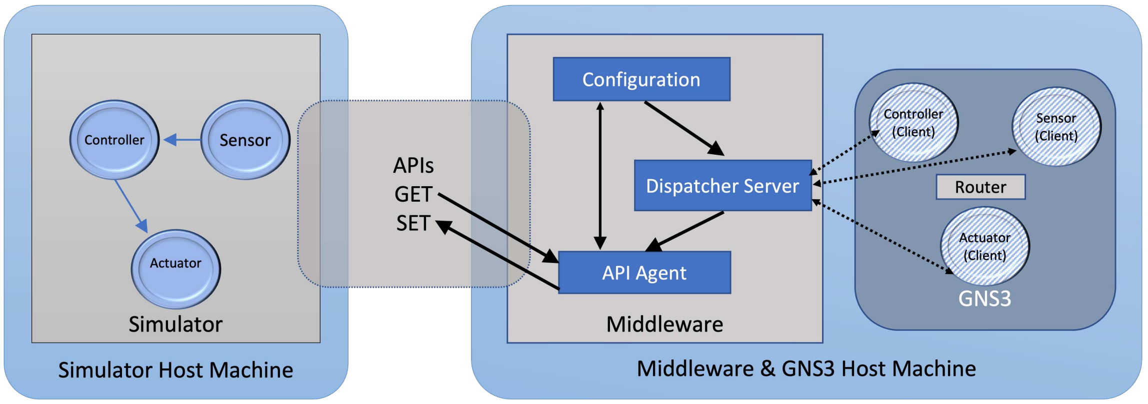


Figure 5.3: Middleware services

2. It also relays data from from one client node to another.
3. The dispatcher server also passes data to API agent that needs to be returned to reactor simulator software.

Experiment Configuration Service: This configuration service holds the information about the identities (id, IP address, port) of the client nodes. It also has information of the type of entities, defined paths etc. which helps create appropriate instructions.It also holds a buffer of instructions that need to be executed by the dispatcher server. It is responsible for creating sets of instructions (E.g, SEND SENSR_1 10.10.10.1 9999 DATA TEMP_1 100) following communication template (<COMMAND NODE_ID DEST_ADDR DEST_PORT DATA MSG>) according to experiment configuration.

5.3 Implementation of Design

According to our design requirement, the dispatcher server program, a component of the middleware program, will be running on the host machine and the client program on each GNS3 node at the virtual layer. So, one of the major tasks of the implementation is configuring the network connection between GNS3 virtual layer and the local host machine. Developing the sub components (dispatcher server, API agent, communication protocols etc.) of the middleware component is another implantation task. In the following, details of each implementation task are explained.

Deployment of GNS3: GNS3 is a third party open source software which allows to create network topology using virtual resources. It is a network simulation software. GNS3 consists of two software components:

1. GNS3-all-in-one software (GUI)
2. GNS3 virtual machine (VM)

The GNS3-all-in-one is the client component of GNS3, a graphical user interface (GUI). After installing the all-in-one program on the local PC (Windows, MAC, or Linux) and it can be used to design topology. This is what one generally sees like the figure 5.4.

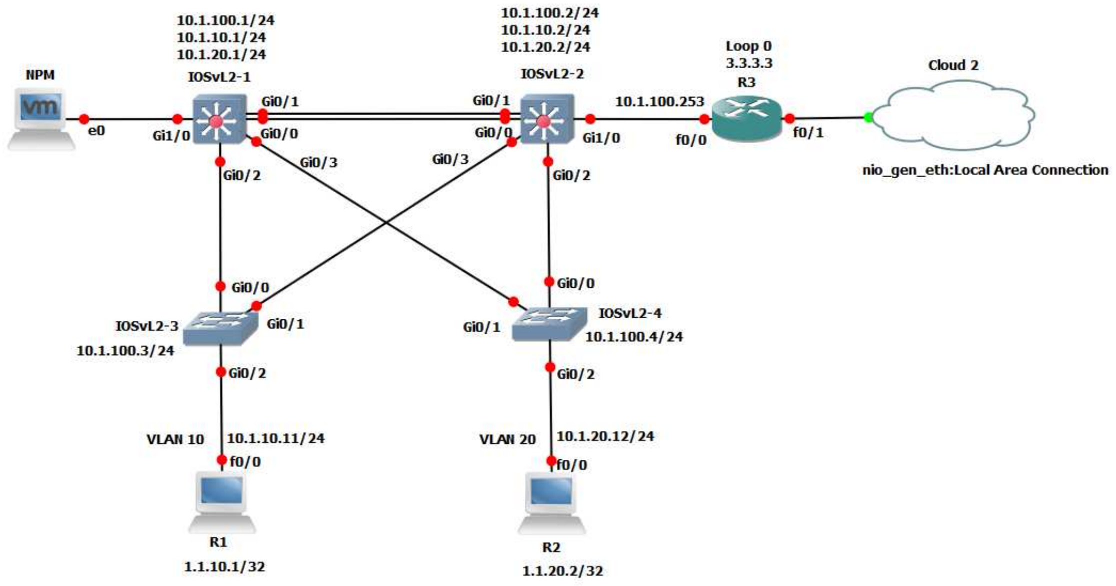


Figure 5.4: Network topology in GNS3 GUI

Devices must be hosted and run by a server process when creating topology in GNS3 using the all-in-one software GUI client. For the server portion of the software, there are few options:

- Local GNS3 server
- Local GNS3 VM
- Remote GNS3 VM

On the same computer where the GNS3 all-in-one software is installed , the local GNS3 server is operated locally. For instance, if a Windows computer is used, Windows is running processes for both the GNS3 GUI and the local GNS3 server. In our case, we are running local GNS3 VM as a server using VMware workstation.

So, there are some prerequisites for making GNS3 work in our case. They are as follows:

1. Windows 10
2. Installing GNS3[5]

[5]Documentation for installing GNS3 on windows, https://docs.gns3.com/docs/getting-started/installation/windows/

3. Installing VMware workstation [6]

4. Installing GNS3 [7]

5. Import router image (C3600)[8]

Network Configuration: As virtual devices or assets on GNS3 need to access the host network so that middleware dispatcher server can perform communication via TCP socket connection, a bridged network between the GNS3 VM (Running on VMware workstation) and the host needs to be established. In bridged networking, a physical network adapter on the host system is connected to the virtual network adapter in the virtual machine (Figure 5.5). In our case, GNS3 VM running on VMware is our specified virtual machines where Hall our virtual devices are hosted. The virtual machine can connect to the host system's LAN via the host network adapter. With both wired and wireless host network adapters, bridged networking is functional.Through the use of bridged networking, the virtual machine is set up to have its own distinct identity on the network, unconnected to the host system and other network resources. The virtual machine is a complete network participant. It can communicate with other computers on the network as if it were a real computer on the network, and it has access to other computers on the network.

The detail of configuring network for windows host machine and GNS3 VM can be found in Appendix A and Appendix B. In Appendix C, the process establishing bridged network incorporating a basic topology on GNS3 GUI has been presented.

API agent: API agent is a program which handles communication with simulator software. It calls SET or GET API to pull or push data from or to the simulator. It is also responsible for creating instructions for dispatcher server to execute according to standardized

[6]Official Website for installing VMware Workstation, https://www.vmware.com/products/workstation-player.html

[7]Documentation for installing GNS3 VM on windows, https://docs.gns3.com/docs/getting-started/setup-wizard-gns3-vm

[8]Find non-official image for router C3600, https://mega.nz/folder/nJR3BTjJ#N5wZsncqDkdKyFQLELU1wQ

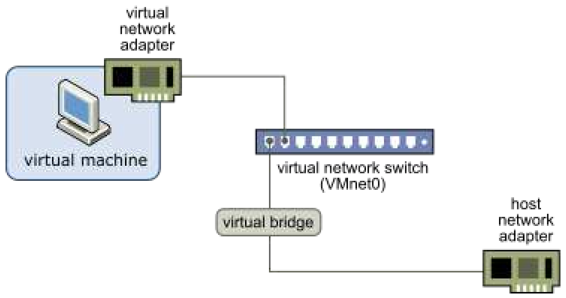


Figure 5.5: Bridged network between VM and host

format of instructions. Suppose, we want to replace S -> C using network simulation on GNS3 (Explained in Figure 5.1). S' -> C' is handled in GNS3. But S -> S' and C' -> C are handled by API agent using GET and SET API respectively. The definition of GET API is `get(connection_to_simulator,var_name)`. Here `var_name` can be any variable from any component of the software simulator. According to our example case, `get(connection_to_simulator,"TEMP_1")` needs to be called and value of that specific temperature will be returned. Then API agent will create appropriate instructions for dispatcher server to dispatch. In case of sending data back to simulator (C'-> C), the `set(connection_to_simulator,var_name,value)` is called. An example instruction for S' -> C' may look like following:

SEND SENSR_1 10.10.10.1 9999 DATA SEND CTRL_1 10.10.10.2 9998 DATA TEMP_1 100

Dispatcher server and node client communication: The dispatcher server is a server program running on the host machine on a specific port. It waits for socket connections with each client node running on GNS3 to be established. Once a client connects to the dispatcher server a separate thread is created to handle the communication. On the other hand, a client program is running on each virtual node on GNS3. And each node sends request for establishing connection to the dispatcher server running at specific address and port.

Dispatcher Server program will have the following major functionalities:

- **handle_client(client_socket, client_address):** For each client, after establishing the connection with the dispatcher server, the `handle_client` method is being executed in a thread. Getting packets from the clients and responding accordingly are handled separately for each client.

- **process_config(file_name):** This method reads the `config` file, parse the instructions and process the instruction accordingly. Suppose, the instruction is:

  SEND SENSR_1 10.10.10.1 9999 DATA SEND CTRL_1 10.10.10.2 9998 DATA TEMP_1 100

  This method will parse it considering the following template:

  <COMMAND NODE_ID DEST_ADDR DEST_PORT DATA MSG>

  The parsed element are following:

  – COMMAND:= SEND
  – NODE_ID:= SENSR_1
  – DEST_ADDR:= 10.10.10.1
  – DEST_PORT:= 9999
  – MSG:= "SEND CTRL_1 10.10.10.2 9998 DATA TEMP_1 100"

  So message "`SEND CTRL_1 10.10.10.2 9998 DATA TEMP_1 100`" will be sent to node `SENSR_1` that is running at `ip address 10.10.10.1` and `port 9999`.

Client program will have the following major functionalities:

receive_messages(client_socket) This method is responsible for receiving message from the dispatcher, parse it (if necessary) and store it in a queue for future processing. Suppose, after receiving the following message:

SEND CTRL_1 10.10.10.2 9998 DATA TEMP_1 100

It will be stored in the queue.

send_messages(client_socket): This method is responsible for creating appropriate message from the queue and sending it to dispatcher server. Suppose, the following content is dequeued first:

```
SEND CTRL_1 10.10.10.2 9998 DATA TEMP_1 100
```

Then this message will be sent to dispatcher server. This method runs on separate thread other than main thread or receive_message thread.

The code-snippet for dispatcher server and client node are included in Appendix D.

All the resources for implementation have been maintained in a github repository.[9]

[9]https://github.com/rafi075/csugw

Chapter 6

Protection of Testbed

In this chapter we address the problem of how to protect the security focused testbed from the unpredictable events that might occur while multiple components/organization/experiments need to collaborate with each other. Our goal is to devise a protection framework based on safe communication between experiments deployed in isolation.

6.1 Threat Model for CPS Testbed

Before delving into our contributions, we would like to explain the threats that a security-focused testbed must guard against. Experiments perform various attacks on different simulated physical models of different devices in the testbed. If an attack goes beyond any experiment run-time environment, it causes severe damage to another experiment, like modifying configuration files, input data sets, output results, and others. The testbed threats can mainly be classified into two groups: *(i) outsider threats* and *(i) insider threats* and are discussed in the following.

6.1.1 Outsider Threats

A testbed is vulnerable to threats that are initiated from outside of it. Malicious actors from the outside world can exploit the testbed to compromise it and gain confidential information from it. Outside attackers can compromise the testbed by exploiting vulnerabilities in the testbed's hardware and software resources. In this work, we assume the testbed is secured from outside threats.

6.1.2 Insider Threats

Insider threats are originated within the testbed itself. Multiple experiment nodes share testbed resources like hardware, software, attack library module, network I/O, and other essential resources while deployed on the testbed and executed at the same time. Without any preventive mechanism, an experiment can access information from another experiment by unauthorized means and can push malicious data into another experiment's memory space. We classify testbed insider threats into three groups: *(i) confidentiality threats*, *(ii) integrity threats*, and *(iii) availability threats*.

Confidentiality threats: Critical Cyber Physical Systems (CCPS) have many critical units to provide services. Since the testbed simulates the actual behaviors of those systems, it can hold notable data about those critical units and proprietary information about the system organization. If a malicious entity gains access of an experiment node, it may also obtain significant architectural information or system vulnerabilities. This reveal of system knowledge may help the malicious entity to plan a more sophisticated attack in the testbed. Moreover, malicious organizations can use that information to gain business profits and defeat competitors in business market competition.

Integrity threats: An erroneous input data or process can introduce faulty results or add bias to the experiment results. An experiment can intentionally or unintentionally violate the data integrity of other experiments, if attack and data isolation are not maintained properly. As experiments share the same testbed resources, if a set of attacks goes beyond a component's memory scope, then an erroneous evaluation may take place.

Availability threats: A malicious experiment can hold the computing resources intentionally for indefinite time period to make the service/resources unavailable for other experiments. In this scenario, other experiments remain in the waiting queue for log time because of resource shortage. Our current scope of proposed method does not deal with this kind of threat.

6.2 Experiment Communication Model

Experiments are deployed as a combination of isolated nodes in the testbed. An isolated node of an experiment cannot communicate or share any data with other experiment nodes using the inter-process communication (IPC). But there are some scenarios where experiments need to exchange information to complete tasks. In this section, we identify and explain two types of communication required for our proposed security framework of communication in the testbed environment. They are: *(i) coordination communication* and *(ii) collaboration communication.*

Suppose, two experiments are active at the same time on the testbed and one experiment needs to access some data that another experiment contains. Any kind of communication that involves two nodes from different experiment is supported by the combined execution of coordination and collaboration communication. The coordination communication model (illustrated in figure 6.1) is occurred first to ensure the availability of resources, approval of the communication from etc. The actual intended communication or data sharing will not start before the coordination completes and the approval is received. Collaboration communication (illustrated in figure 6.2) starts when coordination is complete and nodes have received approval in the completed coordination process.

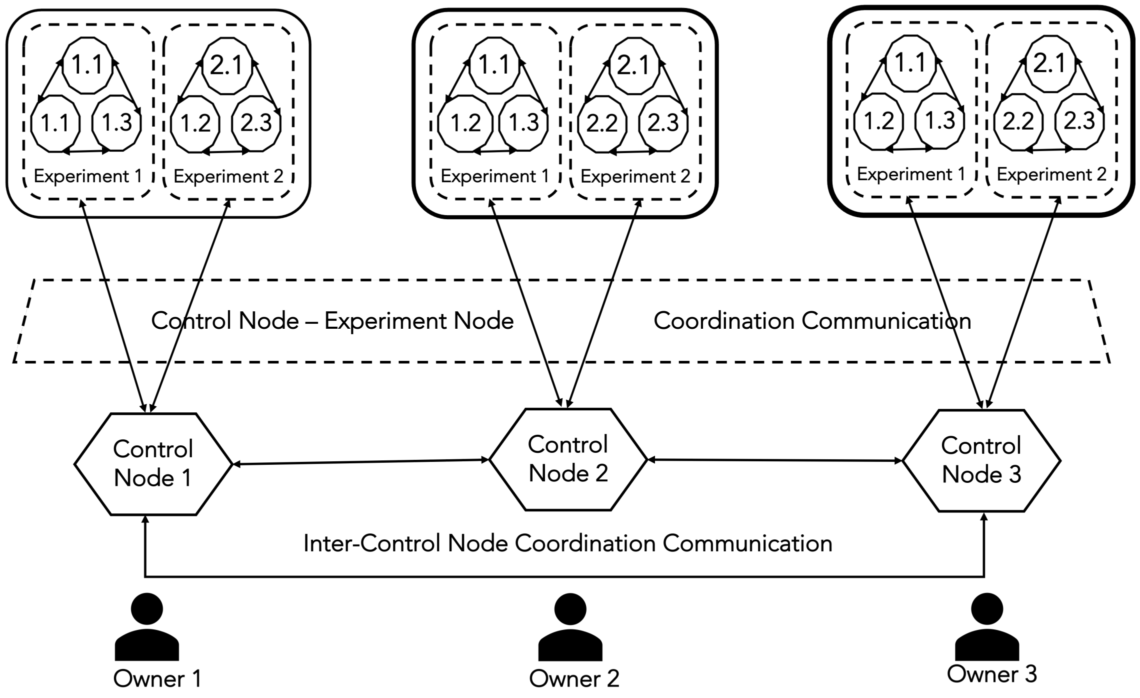


Figure 6.1: Coordination communication.

Coordination communication: A control node performs the main role in this communication model, where it sends predefined control messages to/from other experiment nodes. A control node acts as a coordinator on behalf of a user to initiate, manage, and terminate experiments in the testbed environment. A control node can send/receive coordination information (data access request, broadcast request, node summary, etc.) to/from the experiments it controls or the other control nodes. The control node also expects feedback if it sends messages to other control nodes or experiments. Fig. 6.1 depicts the coordination communication model where two types of coordination communication may occur: (i) control-to-control, (ii) control-to/from-experiment. In coordination communication, no experiment data is shared. Coordination communication should occur first if two experiments need to collaborate (share data).

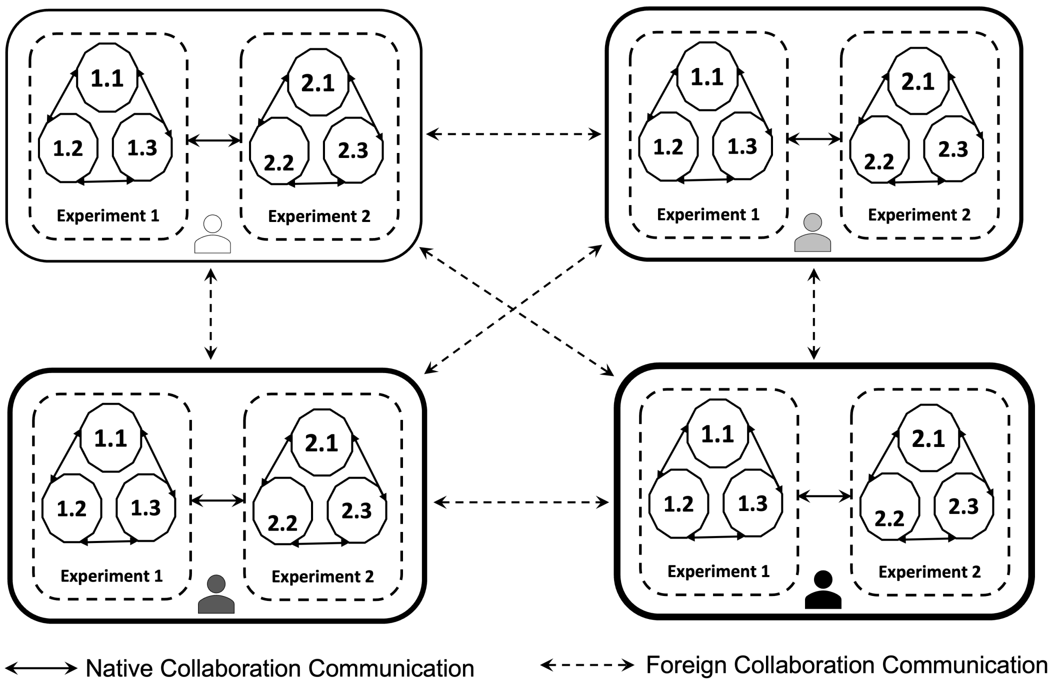

1.1
1.2
1.3
2.1
2.2
2.3
Experiment 1
Experiment 2
Native Collaboration Communication
Foreign Collaboration Communication

Figure 6.2: Collaboration communication.

Collaboration communication: The actual data transfer, labelled as collaboration of experiments, happens between two experiment nodes in this type of communication. No collaboration occurs without executing the coordination communication beforehand. In the collaboration communication, no control node is involved. The solid bi-directional arrows in Fig. 6.2 indicate this native collaboration where experiments under the same control collaborate (data transfer) with each other. In foreign collaboration communication (depicted in the Fig. 6.2 with dotted arrows), experiments from one control can communicate with experiments from another control node.

6.3 Overview of Experiment Execution on Testbed

In this work, we provide a system design for the testbed to provide control over communication among the experiments. Our proposed approach for inter-experiment communication leverages the technology of tuple space in an isolated environment. We assume that the testbed already has the facility of providing virtualization and isolation of nodes.

Experiment initialization: When an owner wants to create experiments, a control node (allocating an isolated node) is instantiated first. A secure communication channel between a user and a control node is established to exchange necessary information. A control node is responsible for managing multiple experiment nodes for its owner. It passes a resources and privileges allocation request containing the necessary node configuration from the owner to the authorization module. After getting the approval, multiple isolated nodes are allocated with a defined networking topology among them. For each initialized node, a tuple space is initialized, which is required for future coordination or collaboration communication. Thus, an experiment is initialized in the testbed.

Inter-experiment secure communication: When two nodes from different experiments (owned by the same or different user) need to communicate, secure communication via tuple space is used. A tuple space is a service that provides a dedicated memory region in the isolated node's local memory space to store data or remove data when required. Only a mapped node itself and the tuple space manager (TSM) can access its memory region and perform an action on it. All the operations with the tuple space by TSM or the node itself are assumed to be secured. In this mechanism, two experiment nodes do not directly communicate with each other; rather, TSM (assumed as trusted) passes information from one tuple space to another.

Authorization of communication: It is reasonable to assume that the communication between all the pairs of nodes from different experiments is not allowed directly. Whether communication between a pair of nodes is allowed or not is defined by the access control policy.

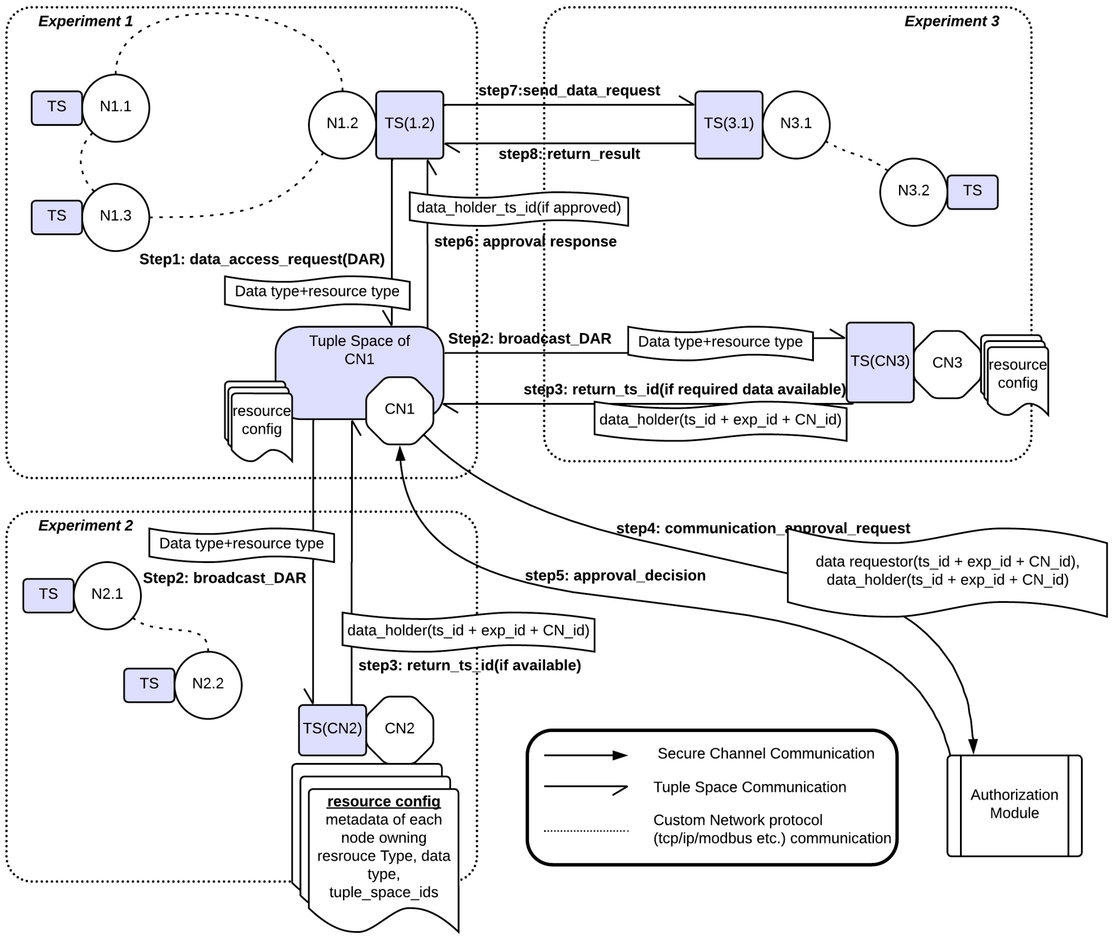
When one node needs to access information held by another one from a different experiment space, a communication request is sent to the authorization module via the control node. The approval decision is sent back to the control node after evaluating the predefined access control policy. The process of approving resource allocation undergoes the same approval procedure.

6.4 Inter-experiment Secure Communications

Three types of communication channels are notably used in the testbed to fulfill the requirements of communication nature. Nodes from the same experiments communicate with each other via standard or specialized networking protocols without prior approval (called intra-experiment communication, illustrated as a dotted line in Fig. 6.3). Communication is performed using the IP address or hostname of the nodes.

The secured communication between the authorization module and any control node (illustrated as a solid arrow in Fig. 6.3) is another type of channel. The third type of communication is between two tuple spaces. Each node (experiment or control) is mapped with a tuple space (illustrated as a rectangular purple region attached to each node). The details of tuple space communication managed by a Tuple Space Manager (TSM) are described in section 6.5. Fig. 6.3 illustrates the step-by-step process of inter-experiment communication involving different channels and related components. The steps are explained in the following:

Send data access request(DAR): The data requester node sends the request information to the control node of its own experiment space (we call it the requester control node) first, leveraging tuple space communication (step 1 from Fig. 6.3). The data requester does not know which node has the specific type of data that the requester wants. It only informs its own control node about the requirements of the specific data type. And in return, it only expects the identity of the data-holder tuple space so that it can start communication with it. No IP address or machine address of any node is disclosed in the whole communication process.



Experiment 1
Experiment 3
Experiment 2
TS
N1.1
N1.2
TS(1.2)
TS
N1.3
step7:send_data_request
step8: return_result
TS(3.1)
N3.1
N3.2
TS
data_holder_ts_id(if approved)
step6: approval response
Step1: data_access_request(DAR)
Data type+resource type
Tuple Space of CN1
CN1
resource config
Step2: broadcast_DAR
Data type+resource type
TS(CN3)
CN3
resource config
step3: return_ts_id(if required data available)
data_holder(ts_id + exp_id + CN_id)
step4: communication_approval_request
data requestor(ts_id + exp_id + CN_id), data_holder(ts_id + exp_id + CN_id)
step5: approval_decision
Data type+resource type
Step2: broadcast_DAR
TS
N2.1
TS
N2.2
data_holder(ts_id + exp_id + CN_id)
step3: return_ts_id(if available)
TS(CN2)
CN2
resource config
metadata of each node owning resrouce Type, data type, tuple_space_ids
Secure Channel Communication
Tuple Space Communication
Custom Network protocol (tcp/ip/modbus etc.) communication
Authorization Module

Figure 6.3: Inter-experiment communication flow diagram.

Broadcast DAR to other control nodes: After receiving the DAR, the control node looks for the identity information of the requested data type from the past communication history that is stored in the *resource config*. A resource config file is maintained at every control node to provide necessary information about tuple space (TS) identities of experiment nodes it controls, experiment identities, mapping information between nodes and experiments, type of data each node holds, past communication history, TS identities of other control nodes, etc. This resource config file is updated from time to time if any event occurs at its own experiment spaces or other control nodes so that the config information remains consistent. Resource config files can also be updated when any previous approval decision is changed. If any approved active tuple space is found from the history as a data holder, no further approval is necessary. The TS id of the data holder is returned to the requester (go to step 6 as illustrated in Fig. 6.3). But if there is no history of prior approval of the same DAR, then a broadcast to all the other control nodes takes place (step 2 from Fig. 6.3). It relays the same message of DAR in this broadcasting phase.

Return data/resource availability information: After getting a broadcast DAR from any other control node, the receiving control node will look for the availability of the data type in its own resource config file. If there is any experiment node that holds the requested type of data, the corresponding control node will find that information in its resource config file and return the TS id of the data holder along with other information (experiment id, control node id, etc.) back to the requester control node. If a receiving control node finds no data availability of the requested type, 'NOT_AVAILABLE' is sent back (step 3 from Fig. 6.3).

Send approval request to the authorization module: Now the requester control node has the information (node TS id, experiment id, etc.) about the data requester and data holder. Using the secured communication channel already established between the control node and the authorization module, an approval request is sent to the authorization module to assess whether the requested data from the data holder can be accessed or not. (step 4 from Fig. 6.3).

Return approval decision: After receiving an approval request from a control node, the authorization module intends to check if the request complies with the access control policies. Details of the authorization module and access control can be found in section 6.6. The approval decision is notified to the tuple space manager to update the *approved_communication_list*. Finally, the approval decision is sent back to the requester control node (step 5 from Fig. 6.3).

Return TS id to requester: Before passing the approval decision to the data requester node, the control node stores this information in the resource config file for future use. If the DAR is approved, the TS id of the data holder node is included in the approval response message. If the DAR is denied or data is not available, DENIED or NOT_AVAILABLE is included in the approval response message. No further communication takes place; this flow terminates here.

Send data request to data holder: If the DAR is approved, the awaited communication via tuple space takes place now. First, a data request message is passed from the requester to the holder via tuple space manager. This message includes the holder's TS id and data type.

Return result to data requester: In response to the data request sent by the requester, the requested result is passed from the holder to the requester.

6.5 Testbed Tuple Space Design for Isolated Experiments

The tuple space model provides a mechanism which allows experiment nodes placed in an isolated environment share data without using any direct communication channel. In the following, we discuss the tuple space manager, tuple space operations, and tuple space transactions.

6.5.1 Tuple Space Manager-TSM

The TSM is a secured and trusted entity that performs data transfer operations from source tuple space to destination tuple space. A secured entity protects data from being modified or revealed to illegitimate subjects. Also, it does not analyze the tuple space content (called tuple) to disclose data to other entities or to learn more about data for itself. It needs to read a tuple from the sender tuple space and add it to the receiver tuple space. There is only one global TSM in the proposed security-focused testbed. The TSM maintains an *approved_communication_list* that gets updated by the authorization module from time to time. The list is essential to crosscheck the authorization status of the incoming request to prevent malicious transactions.

6.5.2 Tuple Space Operation

The proposed framework supports three basic tuple space operations: *(i) write*, *(ii) take*, and *(iii) read* and described in the following.

Write(tuple): This operation provides the functionality to add a tuple within the mapped tuple space. It does not modify the tuple space contents. The sender node and the tuple space manager can execute this operation. The sender node uses this to add the tuple into the tuple space it owns. The TSM executes this operation by adding the tuple in the receiver node's tuple space.

Take(template tuple): This operation is called to execute an associative search for a tuple that matches the template. Once found, the tuple is deleted from the space and then returned to the tuple space owner's run-time memory. Only tuple space owners can use this function to remove tuples from their tuple space. The Tuple Space Manager cannot execute this operation because TSM does not have any rights to remove tuples from any tuple space.

Read(tuple): To read from the sender's tuple space, the TSM executes the read operation. This operation gets a tuple back from sender's tuple space to TSM's own memory space without removing from the source tuple space.

6.5.3 Tuple Space Transaction

This section provides the illustration (depicted in Fig. 6.4) of a tuple space transaction for communication between two isolated nodes. The tuple space manager is the main medium for passing contents(tuple) from sender tuple space to receiver tuple space. The TSM has access to all the tuple spaces whereas the nodes can access their respective ones.

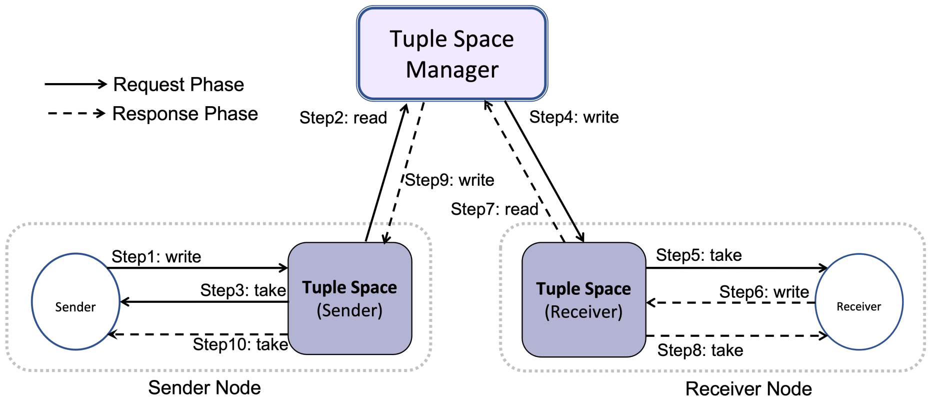


Figure 6.4: Tuple space transaction.

There are two phases in communication: the request phase and the response phase. In the request phase, the sender node sends a message to the receiver node. In the response phase, the receiver node returns a message to the sender node. The returned response can be completely new information or serve as an acknowledgment for the tuple that has just been received. After sending the response message, the communication flow is terminated. Both request and response phase execute the same transaction process to share information which are depicted in the Fig. 6.4. If a node acts as sender in request phase then it is receiver in the response phase and vice versa.

6.6 Authorization Module (AM)

The authorization Module's entire structure and functionalities are explored and illustrated in this section, including the necessary examples. The resources and privilege allocation method are discussed to construct an isolated space for each experiment.

6.6.1 Experiment Resources and Operations

We need to define the resources required and the operations performed by the experiments. The authorization module acts as a reference monitor to control the operations of resources requested by the nodes. An experiment needs multiple hardware and software resources with different privileges to perform operations to complete the scheduled tasks. The control node is responsible for identifying and releasing both resources before an experiment. Some of the resources required by the experimental nodes to complete experiments are network topology configurations, internal memory, disk space, simulated physical models, attack libraries, PLC program control code, proprietary information, snapshots, loggers, status logs, tracers, experimental results, and others.

The experiments in this model perform three operations: *(i) read operation*, *(i) write operation*, and *(i) execute operation*. The *read operation* accesses various files and resources, such as configuration files, input data, programs, and so on. Experiments use *write operation* to write output results, modify configuration files, adjust input parameters, and so on. Finally, the *execute* *operation* allows experiments to run various attack scripts, simulated physical models, and other processes.

6.6.2 Resources and Privileges Allocation

A control node (CN) determines an experiment's required resources and privileges. After selecting the needed resources and rights, CN securely sends the list to the authorization module. In Algorithm 1, the details of resource and privilege allocation processes are given. Both the control node and the authorization module ensure that no experiment is given root permission or excessive resources and rights than required. In addition, the control node can terminate an experiment node after the simulation is completed.

Algorithm 1: Resources and privileges allocation

```
   Input  : A list of resources ($R$) and privileges ($P$), and experiment id $m$.
   Output : IsolatedNode for the experiment $m$ under control node $i$.
1  Initialization:
2  $R \leftarrow \{R_1, R_2, R_3, ........R_r\}$,
3  $P \leftarrow \{P_1, P_2, P_3, ........P_p\}$;
4  $CN_i \rightarrow AM$;
5  AM checks resources' availability and verify resources and privileges legitimacy for the CN;
6  if resource available then
7      AM executes required OS command with $R$ and $P$;
8      $IsolatedNode_{i,m}$ is deployed;
9      AM returns a success message to CN with contained id;
10 else
11     AM returns an error message to CN;
12 end if
```

6.6.3 Relationship between Control Node, Experiments, and Subjects

The distinction between control nodes, experiments, and subjects is fundamental to the authorization module of this work. The relationship between control node, experiments, and subjects is depicted in Fig. 6.5. Usually, each node can perform three operations in the testbed, but in data sharing, only read operation is permitted. The other two operations, write and execute, are not allowed for data sharing.

Control node: Every user has one control node, which acts as a single identity in the testbed environment. An authorized organization has only one control node but multiple control nodes. Violation of this requirement is often the cause of security violations in the proposed system.

Experiments: Each control node may have several experiments associated with it. On the other hand, one experiment must not be mapped to more than one control unit. Therefore, each experiment associated with the control node gains different resources and privileges based on its functionalities.

Subjects: A subject is a program in the system being executed. An experiment can generally spawn several subjects, but each subject is associated with only one experiment. A subject runs with all the privileges of its associated experiment.

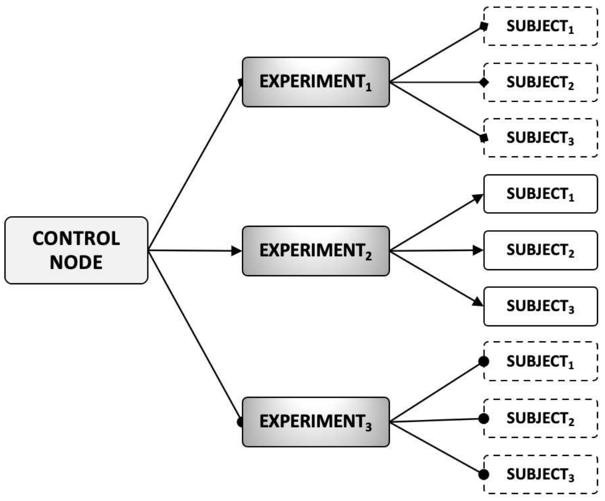


Figure 6.5: Relationship between control node, experiments, and subjects.

6.6.4 Object Classification

To maintain data security and privacy, we classify objects(data) into three main categories. The object classes are the conflict of interest class, agreement class, and open class.

Conflict of interest class: The member objects of this type of class are in a conflict of interest and cannot be shared among their owner.

Agreement class: Objects labelled with this class are in an agreement and can be shared among the agreement signing parties.

Open class: Object data tagged with open class is for all. Any control node can send requests to get the data.

6.6.5 Control Node Identification

Each control node is labeled as a member of the conflict of interest class and the agreement class. The authorization module considers the control node's identity when making a data-sharing decision where the sender and receiver are from different control nodes. Experiments or subjects are not considered. Because experiment nodes from the same control node can share information without regard to security or policy constraints. We avoid experiments or subjects while the access control module approves data sharing to prevent this situation. When a control node is in a conflict of interest class or not in the agreement class, it will not receive data from that class's members.

6.6.6 Security Policy

In this section, we present some access control policies to ensure inter-experiment communication and testbed security and privacy. There are mainly two, native owner and foreign owner, communication scenarios.

Native owner: In this case, we describe data sharing among various experiments when they are all from the same control node. Every node does not need to access data from every node. Experiment nodes must ensure data confidentiality. There are two principles when sharing data among experiments or subjects. Both principles are noted in the following.

Principle 1: *An experiment or a subject can not read data from other experiments or subjects if it violates the confidentiality of the data.*

Principle 2: *An experiment or a subject can read data from other experiments or subjects if they do not violate Principle 1.*

Foreign owner: When there is a data-sharing request where the requester and data holder are from different control nodes, in this case, data sharing can be done if there is no conflict of interest among the experiments. If there is a conflict of interest, the authorization module must not approve data sharing. The conflict of interest issue is raised when the requester and the data holder are from the same conflict of interest class. Data sharing is also possible if there is an agreement between the requester and the data holder. Any control node can access the open class data. A control node puts in a request on behalf of its experiments and subjects. The authorization module depends on the object class and requester control node identity to make the decision. In the following, there are three principles for each type of object.

Principle 3: *A control node can read an object if they are not in the same conflict of interest class.*

Principle 4: *A control node can read an object if they are in the same agreement class.*

Principle 5: *A control node can read any object if objects are in the open class.*

Chapter 7

Conclusion

7.1 Lesson Learned

In this thesis, our target was to design and build a security testbed for CPS to adapt to the new trend of incorporating virtual assets and making the cyber parts accessible for experiments. Moreover, we wanted to ensure that the components and the experiment are protected from insider threats while collaborative ventures frequently occur. To accomplish these goals, firstly, we have analyzed the requirements of two categories: general and security-focused. Then, we tried to identify which supports need to be considered and incorporated into an ideal security-focused testbed. Our design showed that software-based simulators or digital models of physical components integrated with a network simulator leveraging virtualization would give a testbed the maximum trade-off among the necessary security-focused requirements. Again, as we wanted to build a testbed that would protect the authenticity and safe execution of the experiments owned by different organizations, we have incorporated the idea of containerized experiments to provide isolation.

Most of the simulators are owned by proprietary organizations and are of critical national interest. As a result, one of the significant challenges of integrating a software-based simulator within a testbed is inadequate access to simulator software. We accessed a Generic Pressurized Water Reactor simulator fully implemented in software. Nevertheless, these simulators are not designed to perform security experiments to exploit the vulnerabilities at the cyber layer of the system. So, we have modified the design of our ideal testbed by integrating a network simulator called GNS3 with the software simulator of the reactor. This integration allows the creation of a parallel world for the reactor simulator on the GNS3 platform resulting in an exploitable cyber layer that does not compromise the functionality of the original simulator software. It can be highly scalable and flexible when the attached simulator changes its configuration or setup or is replaced by a new one.

We have implemented the design of the middleware component and the applications for communication channels required for passing messages between components and in the parallel world.

To address the protection of the security testbed itself, we have designed a framework for secure communication protocols between isolated spaces dedicated to deploying experiment components. As sometimes multiple experiments from different organizations may need to access each other's resources to reflect upon a well-planned attack simulation or defense mechanism, a four-step process has been proposed to provide a secured environment: data request, coordination, authorization, and collaboration. Furthermore, data transfer via tuple space transaction has been used to preserve the privacy and anonymity of the resources.

As the modern world will incorporate new devices and components into the CPS in the upcoming days, a thousand combinations of devices and components will interact. Moreover, the idea of developing digital twins for physical devices is growing increasingly. Therefore, the testbed we design must have the capability to adapt and integrate with virtual assets or digital components. At the same time, while testing, maintaining those assets' privacy, integrity, and confidentiality is also necessary. Therefore, the proposed testbed's design and implementation architecture contribute to the requirement analysis, developing a prototype, and designing a protection framework for a CPS security testbed.

7.2 Future Work

One of the significant future works would be to design an attack model and deploy it on our GNS3 platform as a module. As there are many tools (like Wireshark) we can integrate with GNS3, it will be convenient for researchers to analyze the impact of the attack experiments. Moreover, We can deploy a data-gathering component into the GNS3 and create a data set to characterize the behavior of a CPS system. This can eventually help to apply different machine learning models to predict the vulnerabilities and risk levels or recommend appropriate defense mechanisms.

Another future work can be to build a generic interface inside the middleware component for integrating any digital twin or virtual component into the testbed. For example, we can plan for including a transformer to turn a digital component into an executable or create an easily extendable API interface to communicate with the digital twin application. More investigation is required to choose the most effective way to do that.

Bibliography

[1] Jianhua Shi, Jiafu Wan, Hehua Yan, and Hui Suo. A survey of cyber-physical systems. In *2011 international conference on wireless communications and signal processing (WCSP)*, pages 1–6. IEEE, 2011.

[2] Ayan Banerjee, Krishna K Venkatasubramanian, Tridib Mukherjee, and Sandeep Kumar S Gupta. Ensuring safety, security, and sustainability of mission-critical cyber–physical systems. *Proceedings of the IEEE*, 100(1):283–299, 2011.

[3] Maria B Line, Inger Anne Tøndel, and Martin G Jaatun. Cyber security challenges in smart grids. In *2011 2nd IEEE PES international conference and exhibition on innovative smart grid technologies*, pages 1–8. IEEE, 2011.

[4] Seungmin Kim, Gyunyoung Heo, Enrico Zio, Jinsoo Shin, and Jae-gu Song. Cyber attack taxonomy for digital environment in nuclear power plants. *Nuclear Engineering and Technology*, 52(5):995–1001, 2020.

[5] Matrix | MITRE ATT&CK®.

[6] An Analysis of the Actual Status of Recent Cyberattacks on Critical Infrastructures : NEC Technical Journal | NEC.

[7] Siddharth Sridhar, Adam Hahn, and Manimaran Govindarasu. Cyber–physical system security for the electric power grid. *Proceedings of the IEEE*, 100(1):210–224, 2011.

[8] Abdallah A Smadi, Babatunde Tobi Ajao, Brian K Johnson, Hangtian Lei, Yacine Chakhchoukh, and Qasem Abu Al-Haija. A comprehensive survey on cyber-physical smart grid testbed architectures: Requirements and challenges. *Electronics*, 10(9):1043, 2021.

[9] Hannes Holm, Martin Karresand, Arne Vidström, and Erik Westring. A survey of industrial control system testbeds. In *Nordic Conference on Secure IT Systems*, pages 11–26. Springer, 2015.

[10] Christos Siaterlis, Andres Perez Garcia, and Béla Genge. On the use of emulab testbeds for scientifically rigorous experiments. *IEEE Communications Surveys & Tutorials*, 15(2):929–942, 2012.

[11] Christos Siaterlis, Bela Genge, and Marc Hohenadel. Epic: A testbed for scientifically rigorous cyber-physical security experimentation. *IEEE Transactions on Emerging Topics in Computing*, 1(2):319–330, 2013.

[12] Nicholas Carriero and David Gelernter. Linda in context. *Communications of the ACM*, 32(4):444–458, 1989.

[13] Aditya Ashok, Adam Hahn, and Manimaran Govindarasu. A cyber-physical security testbed for smart grid: System architecture and studies. In *Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research*, pages 1–1, 2011.

[14] Siavash H. Khajavi, Naser Hossein Motlagh, Alireza Jaribion, Liss C. Werner, and Jan Holmström. Digital twin: Vision, benefits, boundaries, and creation for buildings. *IEEE Access*, 7:147406–147419, 2019.

[15] Marco Bertoni and Alessandro Bertoni. Designing solutions with the product-service systems digital twin: What is now and what is next? *Computers in Industry*, 138:103629, 2022.

[16] Lieven Raes, Philippe Michiels, Thomas Adolphi, Chris Tampere, Athanasios Dalianis, Susie McAleer, and Pavel Kogut. Duet: A framework for building interoperable and trusted digital twins of smart cities. *IEEE Internet Computing*, 26(3):43–50, 2022.

[17] Davide Balzarotti, Paolo Costa, and Gian Pietro Picco. The lights tuple space framework and its customization for context-aware applications. *Web Intelligence and Agent Systems: An International Journal*, 5(2):215–231, 2007.

[18] Poul-Henning Kamp and Robert NM Watson. Jails: Confining the omnipotent root. In *Proceedings of the 2nd International SANE Conference*, volume 43, page 116, 2000.

[19] Pierangela Samarati and Sabrina Capitani de Vimercati. Access control: Policies, models, and mechanisms. In *International School on Foundations of Security Analysis and Design*, pages 137–196. Springer, 2000.

[20] Tamara Becejac, Crystal R. Eppinger, Aditya Ashok, Urmila Agrawal, and James G. O'Brien. PRIME: a real-time cyber-physical systems testbed: from wide-area monitoring, protection, and control prototyping to operator training and beyond. *IET Cyber-Physical Systems: Theory & Applications*, 5(2), July 2020. Institution: Pacific Northwest National Lab. (PNNL), Richland, WA (United States) Number: PNNL-SA-145117 Publisher: Institution of Engineering and Technology (IET).

[21] Mehmet Hazar Cintuglu, Osama A. Mohammed, Kemal Akkaya, and A. Selcuk Uluagac. A survey on smart grid cyber-physical system testbeds. *IEEE Communications Surveys & Tutorials*, 19(1):446–464, 2017.

[22] Ibukun A Oyewumi, Ananth A Jillepalli, Philip Richardson, Mohammad Ashrafuzzaman, Brian K Johnson, Yacine Chakhchoukh, Michael A Haney, Frederick T Sheldon, and Daniel Conte de Leon. Isaac: The idaho cps smart grid cybersecurity testbed. In *2019 IEEE Texas Power and Energy Conference (TPEC)*, pages 1–6. IEEE, 2019.

[23] Ceeman B. Vellaithurai, Saugata S. Biswas, Ren Liu, and Anurag Srivastava. Real Time Modeling and Simulation of Cyber-Power System. In Siddhartha Kumar Khaitan, James D. McCalley, and Chen Ching Liu, editors, *Cyber Physical Systems Approach to Smart Electric Power Grid*, Power Systems, pages 43–74. Springer, Berlin, Heidelberg, 2015.

[24] Mike Hibler, Robert Ricci, Leigh Stoller, Jonathon Duerig, Shashi Guruprasad, Tim Stack, Kirk Webb, and Jay Lepreau. Large-scale virtualization in the emulab network testbed. In *2008 USENIX Annual Technical Conference (USENIX ATC 08)*, 2008.

[25] Kirill Belyaev and Indrakshi Ray. Component-oriented access control for deployment of application services in containerized environments. In *International Conference on Cryptology and Network Security*, pages 383–399. Springer, 2016.

[26] Kirill Belyaev and Indrakshi Ray. On the formalization, design, and implementation of component-oriented access control in lightweight virtualized server environments. *computers & security*, 71:15–35, 2017.

[27] Kenneth Barnes and Briam Johnson. National SCADA Test Bed Substation Automation Evaluation Report. Technical Report INL/EXT-09-15321, 968658, Idaho National Laboratory, October 2009.

[28] Z. Li and R. Kang. Strategy for reliability testing and evaluation of cyber physical systems. In *2015 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, pages 1001–1006, 2015.

[29] Georgia Koutsandria, Reinhard Gentz, Mahdi Jamei, Anna Scaglione, Sean Peisert, and Chuck McParland. A real-time testbed environment for cyber-physical security on the power grid. In *Proceedings of the First ACM Workshop on Cyber-Physical Systems-Security and/or PrivaCy*, CPS-SPC '15, page 67–78, New York, NY, USA, 2015. Association for Computing Machinery.

[30] Joon Sub Kim, Kyeongho Kim, and Moonsu Jang. Cyber-physical battlefield platform for large-scale cybersecurity exercises. *2019 11th International Conference on Cyber Conflict (CyCon)*, 900:1–19, 2019.

[31] Hannes Holm, Martin Karresand, Arne Vidström, and Erik Westring. A survey of industrial control system testbeds. In Sonja Buchegger and Mads Dam, editors, *Secure IT Systems*, pages 11–26, Cham, 2015. Springer International Publishing.

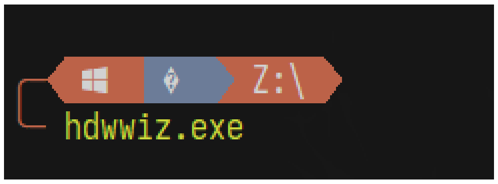
[32] Anton Burtsev, Prashanth Radhakrishnan, Mike Hibler, and Jay Lepreau. Transparent checkpoints of closed distributed systems in emulab. In *Proceedings of the 4th ACM European conference on Computer systems*, pages 173–186, 2009.

[33] Md Rakibul Hasan Talukder, Md Al Amin, and Indrajit Ray. Protecting cyber-physical system testbeds from red-teaming/blue-teaming experiments gone awry. In *Information Security Practice and Experience: 17th International Conference, ISPEC 2022, Taipei, Taiwan, November 23–25, 2022, Proceedings*, page 140–157, Berlin, Heidelberg, 2022. Springer-Verlag.

Appendix A

Windows Network Configuration

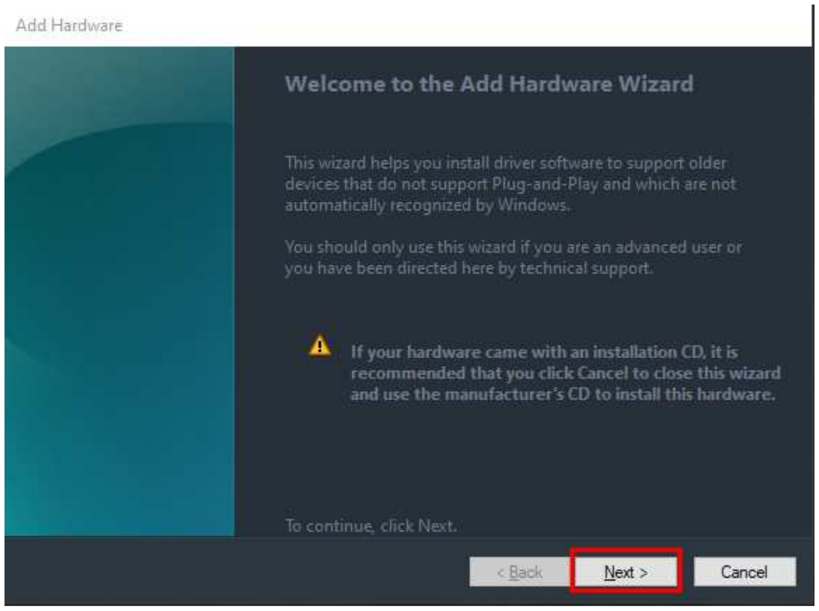
1. Run ***hdwwiz.exe*** in Powershell or CMD (Figure A.1)



```
Z:\
hdwwiz.exe
```

Figure A.1: Opening hardware wizard on windows: step 1

2. Click Next (Figure A.2)



Add Hardware

Welcome to the Add Hardware Wizard

This wizard helps you install driver software to support older devices that do not support Plug-and-Play and which are not automatically recognized by Windows.

You should only use this wizard if you are an advanced user or you have been directed here by technical support.

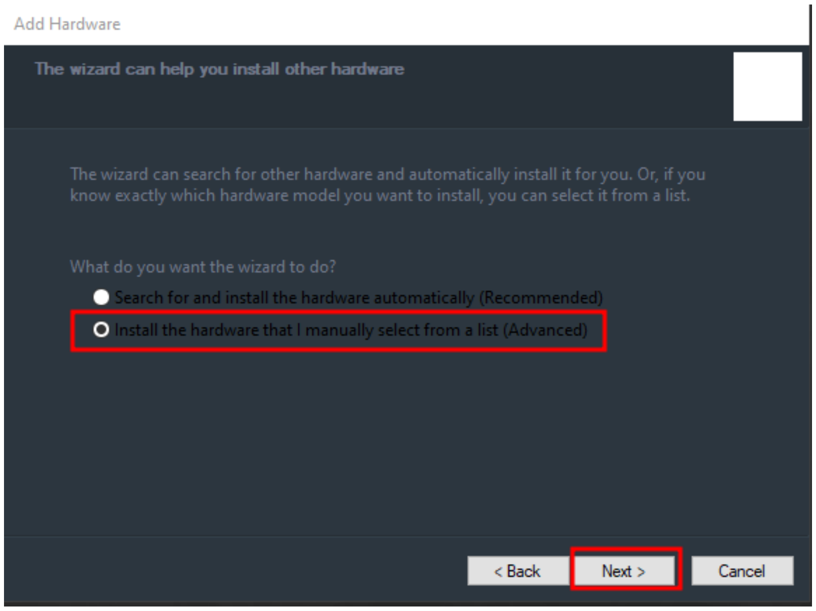
If your hardware came with an installation CD, it is recommended that you click Cancel to close this wizard and use the manufacturer's CD to install this hardware.

To continue, click Next.

Figure A.2: Opening hardware wizard on windows: step 2

3. Choose to install hardware manually then click Next (Figure A.3)

4. Choose “Network adapters”, then click Next (Figure A.4)


Add Hardware

The wizard can help you install other hardware

The wizard can search for other hardware and automatically install it for you. Or, if you know exactly which hardware model you want to install, you can select it from a list.

What do you want the wizard to do?

○ Search for and install the hardware automatically (Recommended)

○ Install the hardware that I manually select from a list (Advanced)

Figure A.3: Manual installation of hardware: step 3

5. Select Microsoft as the manufacturer, and choose the "KM-TEST Loopback Adapter". Click Next twice to install the adapter and Finish. (Figure A.5)

6. Find your new adapter in your network adapter settings (Figure A.6)

7. Right click the adapter (Figure A.7) and do the following:

   (a) Select properties

   (b) Find "Internet Protocol Version 4 (TCP/IPv4)

   (c) Select properties

   (d) Check "Use the following IP address"

   (e) Input an IP address and subnet mask

   (f) Click Ok to confirm settings

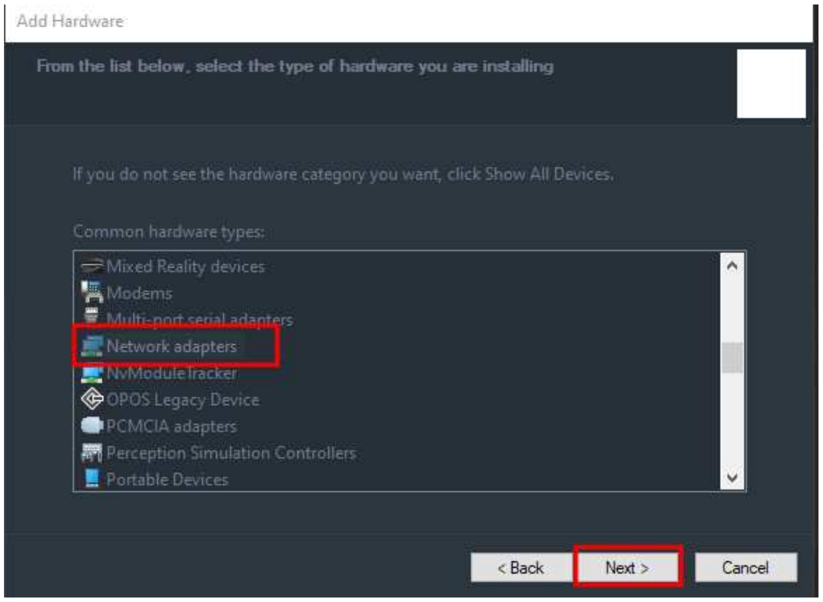

Add Hardware
From the list below, select the type of hardware you are installing
If you do not see the hardware category you want, click Show All Devices.
Common hardware types:
Mixed Reality devices
Modems
Multi-port serial adapters
Network adapters
NvModuleTracker
OPOS Legacy Device
PCMCIA adapters
Perception Simulation Controllers
Portable Devices
< Back
Next >
Cancel

Figure A.4: Choose network adapter as hardware: step 4

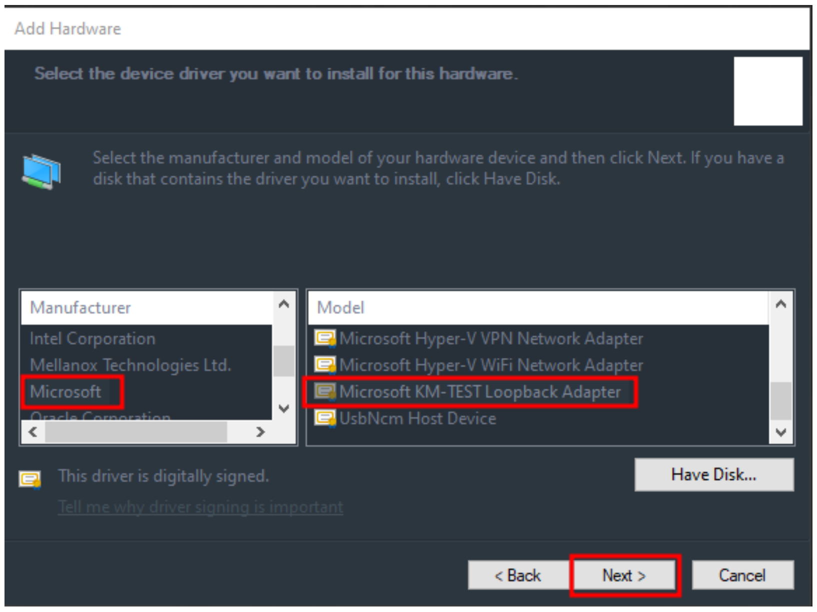
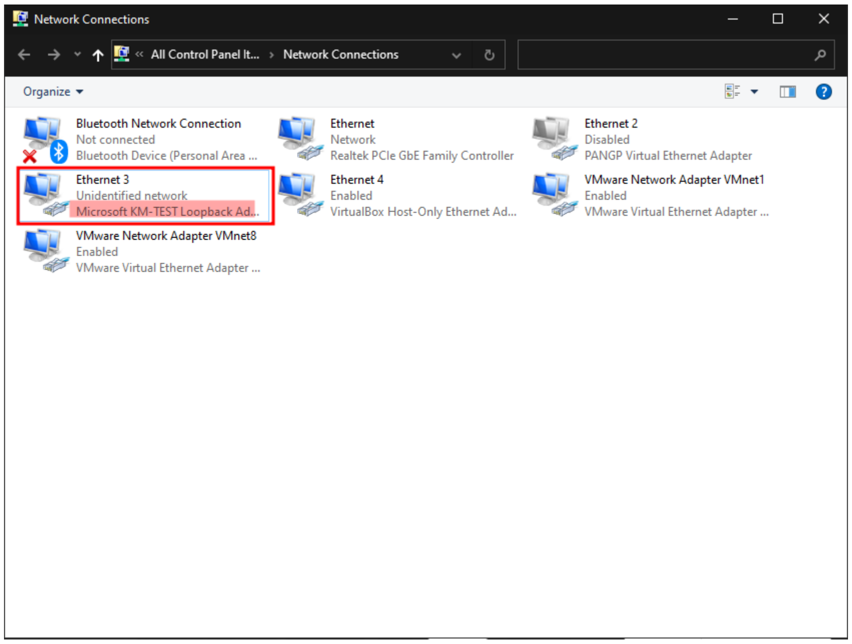

Add Hardware
Select the device driver you want to install for this hardware.
Select the manufacturer and model of your hardware device and then click Next. If you have a disk that contains the driver you want to install, click Have Disk.
Manufacturer
Intel Corporation
Mellanox Technologies Ltd.
Microsoft
Oracle Corporation
Model
Microsoft Hyper-V VPN Network Adapter
Microsoft Hyper-V WiFi Network Adapter
Microsoft KM-TEST Loopback Adapter
UsbNcm Host Device
This driver is digitally signed.
Tell me why driver signing is important
Have Disk...
< Back
Next >
Cancel

Figure A.5: Choose manufacturer and specific network adapter: step 5



Bluetooth Network Connection
Not connected
Bluetooth Device (Personal Area ...

Ethernet
Network
Realtek PCIe GbE Family Controller

Ethernet 2
Disabled
PANGP Virtual Ethernet Adapter

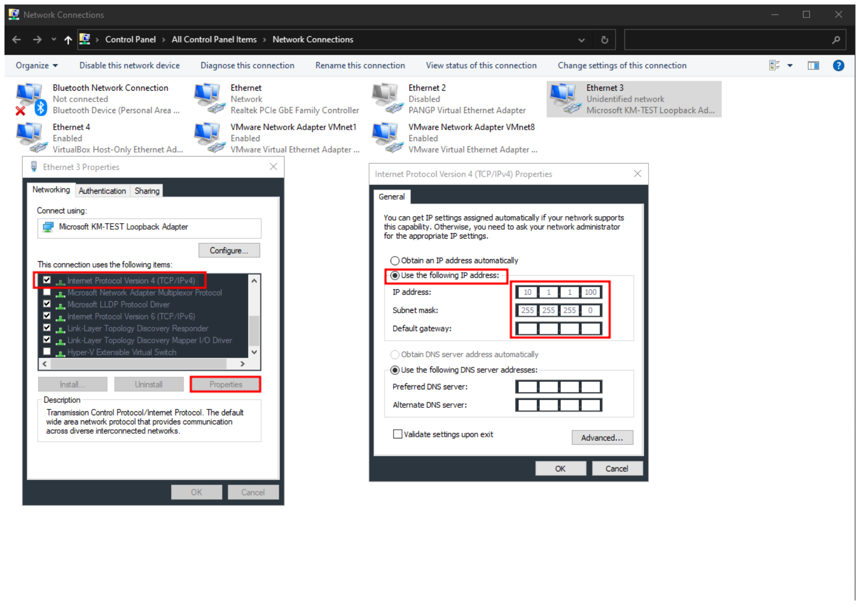
Ethernet 3
Unidentified network
Microsoft KM-TEST Loopback Ad...

Ethernet 4
Enabled
VirtualBox Host-Only Ethernet Ad...

VMware Network Adapter VMnet1
Enabled
VMware Virtual Ethernet Adapter ...

VMware Network Adapter VMnet8
Enabled
VMware Virtual Ethernet Adapter ...

Figure A.6: Find added network adapter in network settings: step 6


Network Connections
Control Panel › All Control Panel Items › Network Connections
Organize
Disable this network device
Diagnose this connection
Rename this connection
View status of this connection
Change settings of this connection
Ethernet 3 Properties
Networking
Authentication
Sharing
Configure...
Install...
Uninstall
Properties
OK
Cancel
Internet Protocol Version 4 (TCP/IPv4) Properties
General
Advanced...
OK
Cancel

Connect using:
Microsoft KM-TEST Loopback Adapter

This connection uses the following items:
- [✓] Internet Protocol Version 4 (TCP/IPv4)
- [ ] Microsoft Network Adapter Multiplexor Protocol
- [✓] Microsoft LLDP Protocol Driver
- [✓] Internet Protocol Version 6 (TCP/IPv6)
- [✓] Link-Layer Topology Discovery Responder
- [✓] Link-Layer Topology Discovery Mapper I/O Driver
- [ ] Hyper-V Extensible Virtual Switch

Description
Transmission Control Protocol/Internet Protocol. The default wide area network protocol that provides communication across diverse interconnected networks.

You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

Obtain an IP address automatically [✓ Use the following IP address:]

IP address:	10 . 1 . 1 . 100
Subnet mask:	255 . 255 . 255 . 0
Default gateway:	

Obtain DNS server address automatically [✓ Use the following DNS server addresses:]

Preferred DNS server:	
Alternate DNS server:	

[ ] Validate settings upon exit

Figure A.7: Configure the adapter with specification of protocols and IP address: step 7

Appendix B

VMware Workstation Network Settings

1. Open VMware Workstation.

2. Open the “GNS3 VM” settings. (Figure B.1)

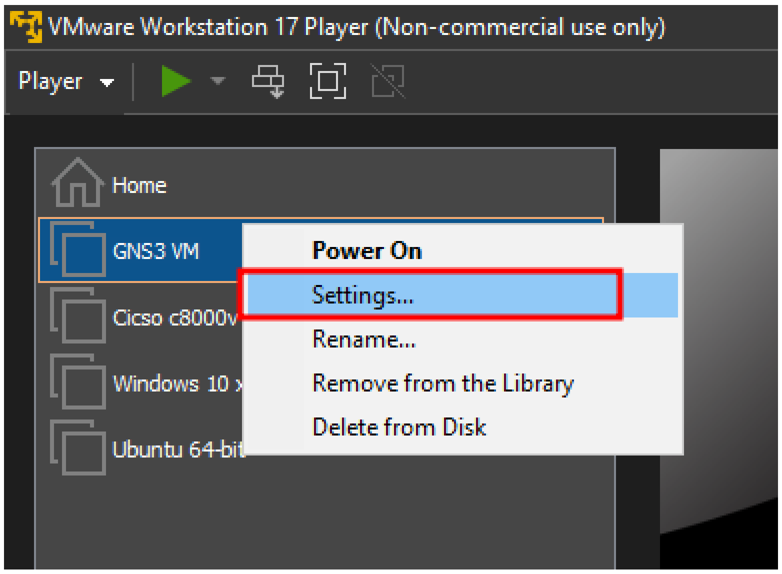
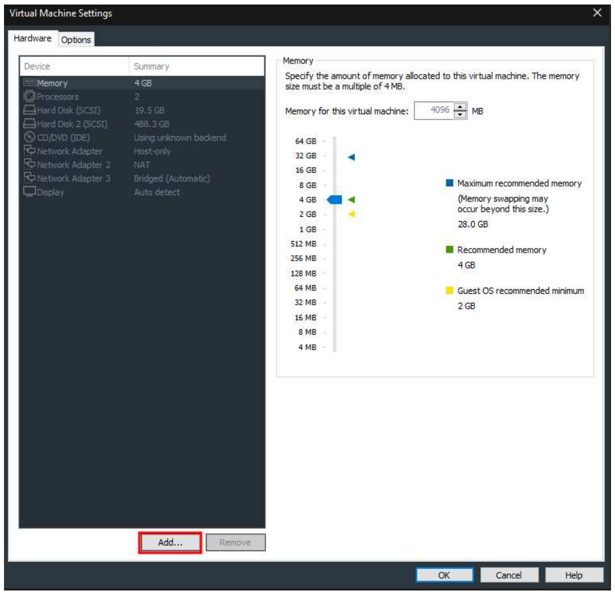


Figure B.1: Opening GNS3 VM settings: step 2

3. Add a new hardware component and select “Network Adapter”. (Figure B.2)

4. Select “Network Adapter 3”, set it to “Bridged” mode, and click “Configure Adapters”. (Figure B.3)

5. Select the loopback adapter we created in the previous section.(Figure B.4)



Device	Summary
Memory	4 GB
Processors	2
Hard Disk (SCSI)	19.5 GB
Hard Disk 2 (SCSI)	488.3 GB
CD/DVD (IDE)	Using unknown backend
Network Adapter	Host-only
Network Adapter 2	NAT
Network Adapter 3	Bridged (Automatic)
Display	Auto detect

Memory

Specify the amount of memory allocated to this virtual machine. The memory size must be a multiple of 4 MB.

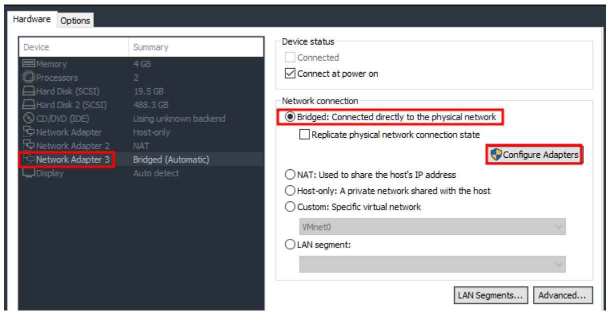
Memory for this virtual machine: 4096 MB

🟦 Maximum recommended memory (Memory swapping may occur beyond this size.) 28.0 GB

🟩 Recommended memory 4 GB

🟨 Guest OS recommended minimum 2 GB

Figure B.2: Adding new hardware component: step 3



Device	Summary
Memory	4 GB
Processors	2
Hard Disk (SCSI)	19.5 GB
Hard Disk 2 (SCSI)	488.3 GB
CD/DVD (IDE)	Using unknown backend
Network Adapter	Host-only
Network Adapter 2	NAT
Network Adapter 3	Bridged (Automatic)
Display	Auto detect

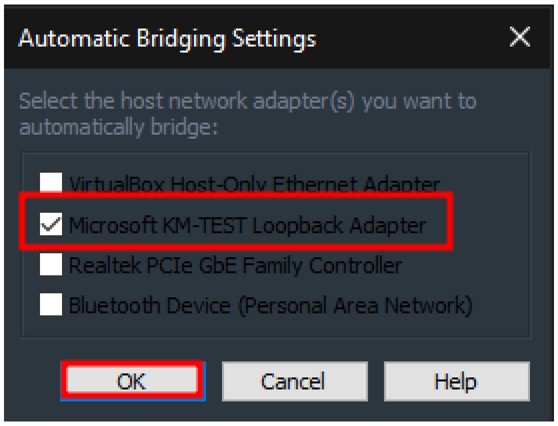
Device status

- ☐ Connected
- ☑ Connect at power on

Network connection

- [✓ Bridged: Connected directly to the physical network]
  - ☐ Replicate physical network connection state
- NAT: Used to share the host's IP address
- Host-only: A private network shared with the host
- Custom: Specific virtual network
  - VMnet0
- LAN segment:

Figure B.3: Setting to ’Bridged’ mode: step 4



Select the host network adapter(s) you want to automatically bridge:

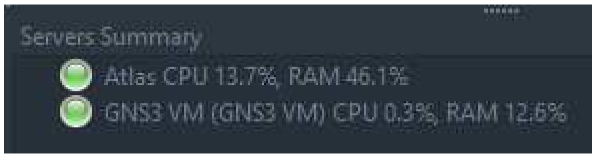
- ☐ VirtualBox Host-Only Ethernet Adapter
- ☑ Microsoft KM-TEST Loopback Adapter
- ☐ Realtek PCIe GbE Family Controller
- ☐ Bluetooth Device (Personal Area Network)

Figure B.4: Selecting loopback adapter: step 5

Appendix C

GNS3 Configuration

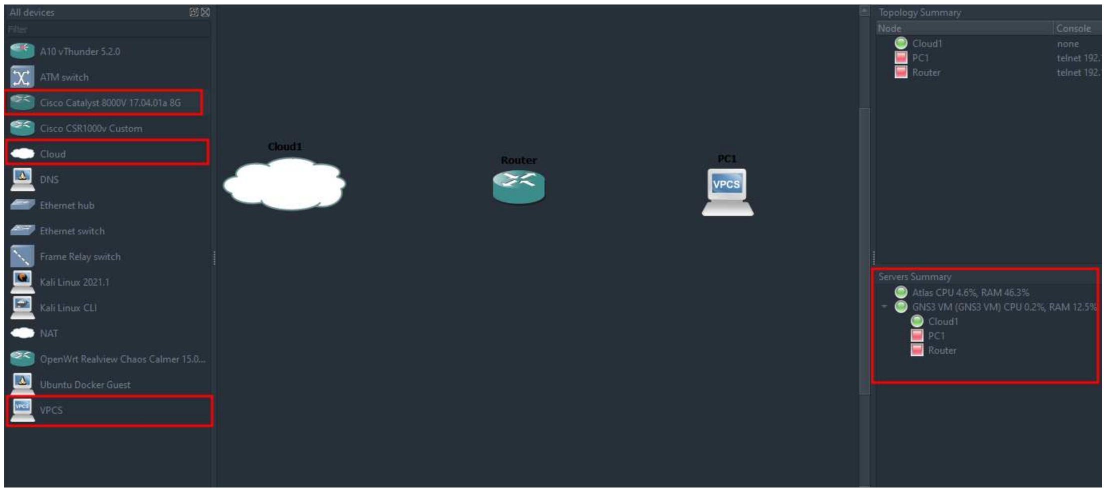
1. Create a new project and wait for both servers to be running.(Figure C.1)



```
Servers Summary
  Atlas CPU 13.7%, RAM 46.1%
  GNS3 VM (GNS3 VM) CPU 0.3%, RAM 12.6%
```

Figure C.1: Creating new GNS3 project: step 1

2. Initialize a Cloud, Router, and PC. Have them run on the VM.(Figure C.2)



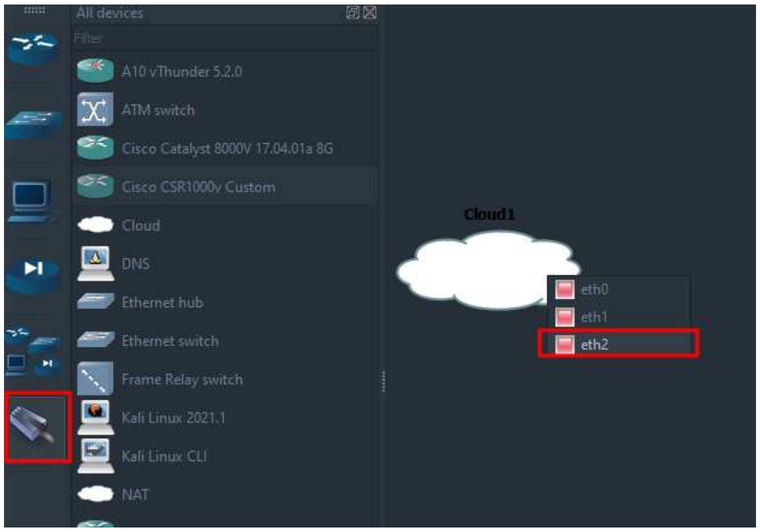
```
Topology Summary
Node        Console
  Cloud1    none
  PC1       telnet 192.
  Router    telnet 192.

Servers Summary
  Atlas CPU 4.6%, RAM 46.3%
  GNS3 VM (GNS3 VM) CPU 0.2%, RAM 12.5%
    Cloud1
    PC1
    Router
```

Figure C.2: Initializing necessary virtual appliances: step 2

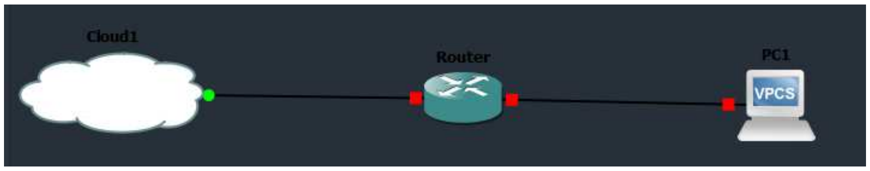
3. Add a link to the cloud using the third network adapter. Then connect all the devices. The subsequent interfaces do not matter.(Figure C.3)

4. Start the router and open a console.(Figure C.4)

5. Enter the following commands in order:

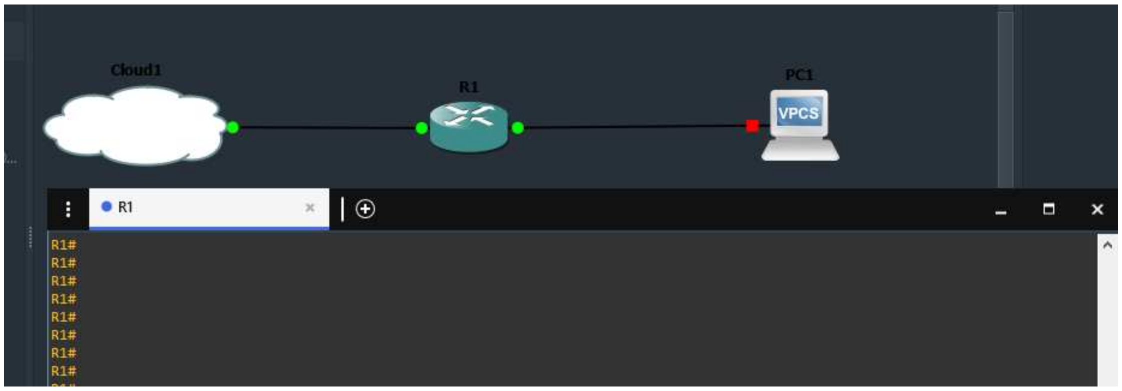


(a) Adding a link to the cloud



(b) Connecting all the devices

Figure C.3: Connecting all the devices via a cloud appliance that works as an interface to connect with the host machine.

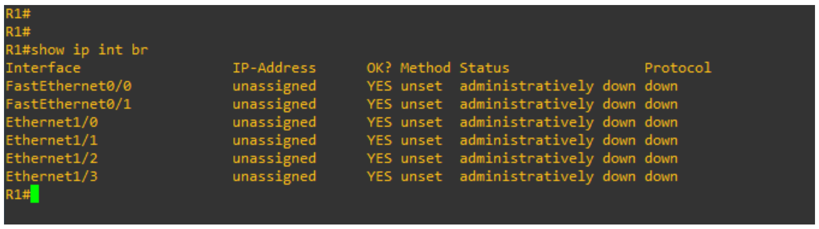


```
R1#
R1#
R1#
R1#
R1#
R1#
R1#
R1#
```

Figure C.4: Starting router and opening the console

Important: These commands will differ depending on the router other than specified one here. These commands are specific to Cisco routers. If anyone would like to follow exactly as what we have done, we are using the c3600 router which can be found in the github project[10]*. This is not an official image host, so reliability of using it is not ensured.*

(a) `show ip int br` (This allows you to see the interfaces on the router. We will use "FastEthernet0/0" like Figure C.5)



```
R1#
R1#
R1#show ip int br
Interface                  IP-Address      OK? Method Status                Protocol
FastEthernet0/0            unassigned      YES unset  administratively down down
FastEthernet0/1            unassigned      YES unset  administratively down down
Ethernet1/0                unassigned      YES unset  administratively down down
Ethernet1/1                unassigned      YES unset  administratively down down
Ethernet1/2                unassigned      YES unset  administratively down down
Ethernet1/3                unassigned      YES unset  administratively down down
R1#
```

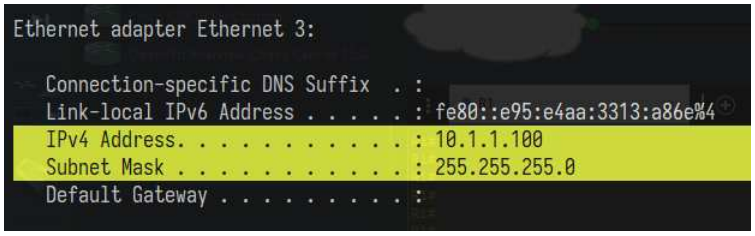
Figure C.5: Showing router interfaces

(b) `conf t`

(c) `int fa0/0`

(d) `ip add 10.1.1.101 255.255.255.0` (This IP is in the range of the loopback interface as shown by ipconfig in Figure C.6)

[10]Our github project csugw, https://github.com/rafi075/csugw

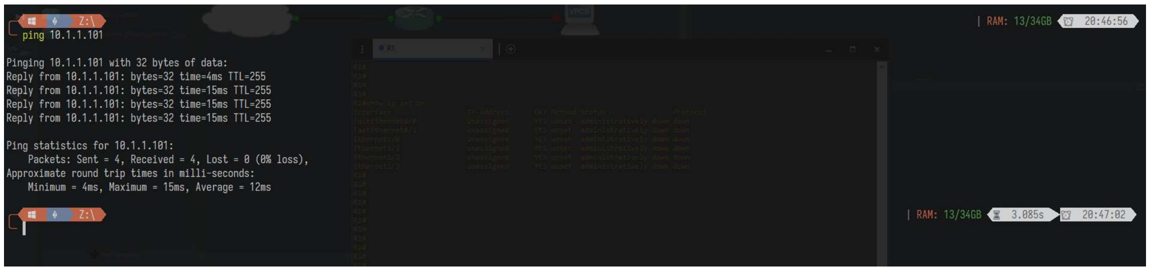


```
Ethernet adapter Ethernet 3:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::e95:e4aa:3313:a86e%4
   IPv4 Address. . . . . . . . . . . : 10.1.1.100
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :
```

Figure C.6: Showing ip address

(e) `no shut`

6. Testing: Now we are able to ping this virtual router from the local computer. Thus connectivity between the host and virtual router is verified. (Figure C.7)



```
Z:\                                                            | RAM: 13/34GB  20:46:56
ping 10.1.1.101

Pinging 10.1.1.101 with 32 bytes of data:
Reply from 10.1.1.101: bytes=32 time=4ms TTL=255
Reply from 10.1.1.101: bytes=32 time=15ms TTL=255
Reply from 10.1.1.101: bytes=32 time=15ms TTL=255
Reply from 10.1.1.101: bytes=32 time=15ms TTL=255

Ping statistics for 10.1.1.101:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 4ms, Maximum = 15ms, Average = 12ms

Z:\                                                | RAM: 13/34GB  3.085s  20:47:02
```

Figure C.7: Verifying the connectivity by Pinging

Appendix D

Server-client communication code

D.1 dispatcher-server.py

```python
import socket
import threading

# Define the IP address and port number that the server will listen
↪ on
SERVER_ADDRESS = ('127.0.0.1', 5000)

# Define a list to store all connected clients
clients = []
ins = []

# Define a function to handle incoming client connections
def handle_client(client_socket, client_address):
    print(f'Client {client_address} connected')

    # Add the client to the list of connected clients
    clients.append((client_socket, client_address))

    # Continuously read incoming messages from the client
    while True:
        data = client_socket.recv(1024).decode()
        if not data:
            break

        # Parse the command and message from the incoming data
        command, message = data.split(':', 1)

        # If the command is SEND, relay the message to all other
         ↪  connected clients
        if command == 'SEND':
            for c, addr in clients:
                if c != client_socket:
                    c.send(f'RECEIVE:{message}'.encode())

        # If the command is SENDTO, find the specified client and
         ↪  send the message to them
        elif command == 'SENDTO':
            to_client_address, message = message.split(':', 1)
            for c, addr in clients:
                if addr == to_client_address:
                    c.send(f'RECEIVE:{message}'.encode())
                    break

    # Remove the client from the list of connected clients
    clients.remove((client_socket, client_address))
    client_socket.close()
    print(f'Client {client_address} disconnected')

def get_client(client_id):

    conn = 0
    addr = 0
    while True:
        if(len(clients) >= client_id):
            conn, addr = clients[client_id - 1]
            break

    return conn,addr

def read_config():
    # using readlines()
    count = 0

    with open("config.txt") as fp:
        Lines = fp.readlines()
        for line in Lines:
            count += 1
            print("Line{}: {}".format(count, line.strip()))
            command,data = line.split(':',1)

            if command == 'SENDTO':
                client_id,message = data.split(':',1)
                client_conn,addr = get_client(int(client_id))
                if client_conn != 0 and addr != 0 :
                    print(f'Send msg {message} to Client {addr}')
                    client_conn.send(f'RECEIVE:{message}'.encode())
                else:
                    print("client not found")
            else:
                print("Command is not recognized")



# Create a socket for the server and start listening for incoming
 ↪  connections
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind(SERVER_ADDRESS)
server_socket.listen()

print(f'Server listening on {SERVER_ADDRESS}')

threading.Thread(target=read_config).start()

# Continuously accept incoming client connections
while True:
    client_socket, client_address = server_socket.accept()
    threading.Thread(target=handle_client, args=(client_socket,
     ↪  client_address)).start()
```

D.2 node-client.py

```
import socket
import threading

# Define the IP address and port number of the server
SERVER_ADDRESS = ('127.0.0.1', 5000)

```

```python
# Define a function to continuously read user input and send messages to the server
def send_messages(client_socket):
    while True:
        message = input()
        client_socket.send(f'SEND:{message}'.encode())

# Define a function to continuously read incoming messages from the server and print them to the console
def receive_messages(client_socket):
    while True:
        data = client_socket.recv(1024).decode()
        if not data:
            break
        command, message = data.split(':', 1)
        if command == 'RECEIVE':
            print(message)

# Connect to the server and start sending and receiving messages
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect(SERVER_ADDRESS)

threading.Thread(target=send_messages, args=(client_socket,)).start()
threading.Thread(target=receive_messages, args=(client_socket,)).start()
```