CSCI 370 Final Report

NREL 1: SOAP Topology

Steven Dillard
Jackson Pow
Colin Myers

December 6, 2024

CSCI 370 Fall 2024

Mr. Caleb Bartel

Table 1: Revision history

Revision	Date	Comments
New	8/22	Made the document and filled in basic information
Rev – 2	8/31	Assignment 1 Requirements
Rev – 3	9/1	Assignment 1 Requirements Revisions
Rev – 4	9/15	Assignment 2 Requirements
Rev – 5	10/17	Assignment 3 Requirements
Rev – 6	11/10	Design Working Doc Requirements
Rev – 7	11/20	Formatting
Rev – 8	12/6	Feedback & Final Touches

Table of Contents

I. Introduction.	4
II. Functional Requirements.	5
III. Non-Functional Requirements.	6
IV. Risks.	7
V. Definition of Done.	8
VI. System Architecture.	9, 10
VII. Technical Design Issues	11
VIII. Software Test and Quality.	12
IX. Project Ethical Considerations.	13
X. Project Completion Status.	14, 15
XI. Future Work.	16
XII. Lessons Learned.	17, 18
XIII. Acknowledgments.	19
IVX. Team Profile.	20
Appendix A – Key Terms.	21

I. Introduction

Virtualization technology has become a cornerstone of modern IT infrastructure, enabling efficient resource utilization, scalability, and flexibility in network design. This project, proposed by NREL, focuses on designing a topology for a "scepter on a platter" technology, leveraging the capabilities of virtualization repositories such as Phenix and Minimega. The goal is to create a network of virtual machines that can effectively simulate real devices, while meeting both functional and non-functional requirements. By simulating this network with virtual machines, NREL can simulate attacks on the devices in a safe environment and not have to worry about breaking expensive hardware that is being used in the power grid. Phenix provides robust management of network configurations, while Minimega offers a lightweight platform for deploying and managing virtual machine clusters. Together, these tools enable the creation of a dynamic, scalable, and secure network topology tailored to the needs of this project. This document outlines the requirements, risks, and criteria for successful completion, guiding the design and implementation of the virtual machine network.

II. Functional Requirements

Network Topology Design:

- Create a virtual machine network topology that supports multiple interconnected VMs.
- Define clear communication pathways between VMs, including support for necessary protocols.
- Incorporate Scepter on a Platter capabilities to allow for ease of setup and sharing of topologies

Integration with Phenix and Minimega:

- Utilize Phenix for advanced network configuration and management.
- Use Minimega as the primary platform for deploying and managing VMs.
- Ensure seamless integration between Phenix and Minimega with defined configuration steps.
- Provide documentation for errors encountered with the communication between Phenix and Minimega

Scalability:

- Support the addition of more virtual machines without requiring major redesigns.
- Ensure the network can handle increased loads efficiently.

III. Non-Functional Requirements

Usability:

- Develop a user-friendly interface for network administrators.
- Ensure ease of use in deploying, configuring, and managing the virtual machine network.
- Provide a highly detailed documentation to assist users in setting up the scepter technology on their own
- Allow for the setting of different protocols for specific devices, such as S7 Comms and OTSIM

Maintainability:

- Ensure the network can be easily changed and adapted to different scenarios with minimal disruption.
- Provide documentation and clear procedures for ongoing maintenance.

IV. Risks

Designing a complex virtual machine network topology comes with several risks that must be carefully managed. One of the primary risks is the potential for integration issues between Phenix and Minimega, as these tools must work seamlessly together to achieve the desired functionality. Misconfigurations or incompatibilities could lead to an incorrectly configured network of virtual machines. Another significant risk is the reliance on virtualization, which, while offering many benefits, also introduces potential points of failure, such as limited resources being shared among virtual machines. Scalability challenges also pose a risk; if the network is not designed with sufficient foresight, it may struggle to handle increased load or additional virtual machines. To mitigate these risks, the project will include thorough testing, regular audits, and the implementation of best practices in both network and virtualization management.

V. Definition of Done

The project will be considered complete when the designed topology meets all defined functional and non-functional requirements and has successfully passed all stages of testing and validation. This includes achieving seamless integration of Phenix and Minimega, with all virtual machines operating within the specified performance parameters. The network should be fully operational, secure, and capable of scaling as needed, with all monitoring and management tools in place and functioning correctly. A comprehensive set of documentation, including user guides, configuration details, and maintenance procedures, must also be provided. Reach goals include a demonstration of the network's resilience to potential risks, such as failure scenarios and security breaches, confirming that appropriate mitigation strategies are in place. Once these conditions are met, and the stakeholders have reviewed and signed off on the deliverables, the project will be officially completed.

VI. System Architecture

Figure 1 illustrates the overarching architecture of the network we are designing. The initial deliverable for this project is a YAML file that specifies the desired network configuration and operational parameters. This YAML file is used to configure **Phenix**, a tool developed by Sandia Labs, which enables the simulation of network topologies. Phenix allows us to model and visualize the network's structure and behavior before actual implementation.

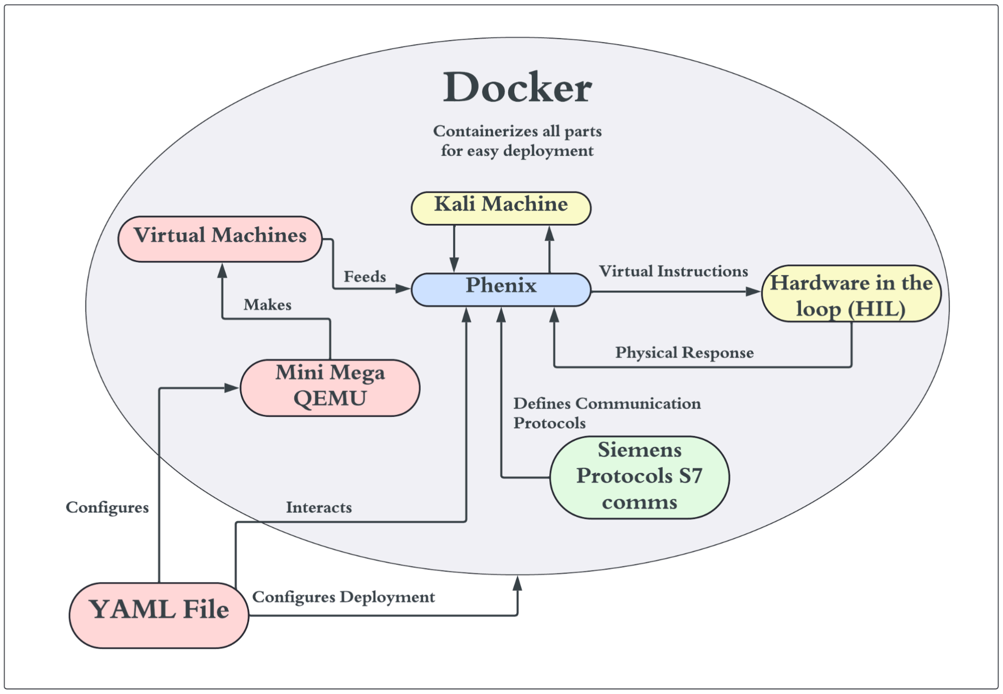

Figure 1. Architecture Diagram

To simulate a large-scale network environment, we employ **MiniMega**, another tool from Sandia Labs. MiniMega is responsible for creating and managing virtual machines (VMs) on a large scale. These VMs are integrated with **QEMU**, which acts as the central processing unit of the network design. QEMU is a versatile emulator that facilitates the execution and management of these virtual machines, providing the core functionality required for our network simulation.

An extra component of our network design is the integration of **Hardware In the Loop (HIL)**. This involves incorporating a real physical device, specifically a **Programmable Logic Controller (PLC)**, into the network. The PLC will be configured to interact with the network of virtual machines. Communication between the PLC and the VMs is managed using the **Siemens S7 protocol**, a standard protocol for PLC communication that ensures seamless data exchange and control.

Figure 2 shows the resulting network after all of the tools are communicating with each other and the topology is set up with the hardware in the loop.

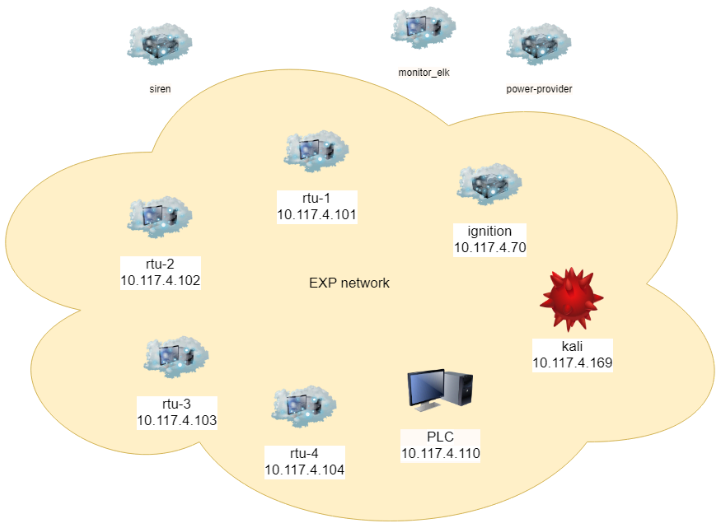

Figure 2. Network Topology

As a stretch goal of our project, we aim to develop a **Kali Linux** machine designed to simulate network attacks. Kali Linux is a well-known penetration testing platform that will help us assess the network's security posture by generating and testing various attack scenarios.

This design approach allows us to create a robust and dynamic network simulation environment, enabling thorough testing and evaluation of both network performance and security.

VII. Technical Design Issues

Unresolved:

- Mini Mega was not configured on start up so additional commands needed to be run.
- Possible permission miscommunication between Mini Mega and Phenix causing running experiments to fail.
- Ignition implementation.

Resolved:

- Issues setting up Phenix locally due to mounting issues
- Changed YAML compose file to accommodate docker volumes which allowed build to complete
- Phenix runs on localhost port 3000 by default which, on Windows, is sometimes used by OneDrive preventing Phenix from running. Solved by using the -listen-endpoint flag to set it manually to 4000. (*Needed to change docker settings to allow containers to run functions over ports.)
- * Namespace error was fixed by including the .qc2 file built by Phenix in the Minimega path

VIII. Software Test and Quality

Our project is unique because the focus is on the configuration of Phenix, which involves significantly less coding compared to typical software development initiatives. Instead of writing extensive code, our primary objective is to tailor Phenix's settings to align with the requirements of our client. Therefore, testing will be more geared towards ensuring that the implementation is not only effective but also suited for the client’s needs.

Given this focus on configuration rather than traditional coding practices, we face the challenge of not being able to utilize conventional testing methodologies to validate our work. In many software projects, developers rely on unit tests and integration tests to confirm that the code performs as intended. However, in our case, the nature of the work means that such tests are not feasible.

To address this challenge, we implement a quality assurance plan that includes elements suited to our project context. While typical software quality elements such as unit testing, user acceptance testing, and code reviews may not apply directly due to our configuration-centric approach, we ensure quality through tailored methods. For example, we utilize Phenix’s built-in testing tool, Scorch. Scorch systematically evaluates each device within the network topology, checking that each device is configured according to the specifications outlined in the YAML file, thereby ensuring that no configuration issues arise during the build process. These features include configurations such as: ip address, operating system, and protocol that the device uses as part of the network.

Additionally, Scorch verifies the integrity of connections throughout the network. It ensures that all connections are established correctly, which is vital for maintaining seamless communication between devices. This capability addresses aspects of defect detection and verification, helping to maintain high quality in the deployment of the configuration.

While our project doesn't align perfectly with conventional software quality techniques such as code audits or static analysis, the combination of configuration validation and connectivity checks provided by Scorch significantly enhances our confidence in the integrity of the network topology, effectively ensuring that our approach to software quality meets the specific needs of our project.

IX. Project Ethical Considerations

Our project involves several critical ethical considerations. Two principles that are especially pertinent to the development of our product are:

- ACM Code of Ethics Principle 1.2: Avoid harm
- IEEE Code of Ethics Principle 7: Work to improve your understanding of technology, its applications, and its consequences.

Principle 1.2 is crucial because it emphasizes the responsibility to minimize harm to users and society. Given that we are using open-source tools, we must ensure that our implementations do not introduce vulnerabilities or misuse these resources. The potential for harm increases when we consider the cybersecurity implications of our work, as weaknesses in our configurations could be exploited by malicious actors.

Principle 7 underscores the importance of continuous learning and understanding the broader implications of our technology. It is essential that we remain aware of how our configurations may affect security and user experience, ensuring that we are making informed decisions based on the latest knowledge and best practices.

Additionally, some principles are at greater risk of being violated, particularly:

- ACM Code of Ethics Principle 3.1: Ensure that the public good is the central concern
- IEEE Code of Ethics Principle 1: Accept responsibility in making decisions consistent with the safety, health, and welfare of the public.

Violating these principles could result in negative impacts, such as exposing vulnerabilities that may lead to security breaches or undermining public trust in technology. The ramifications could extend to financial loss for organizations or even threats to public safety if exploited.

This highlights the imperative to approach our ethical responsibilities with seriousness, ensuring our project not only contributes positively to the community but also safeguards against potential risks. By remaining vigilant about these ethical dimensions and adhering to ACM and IEEE principles, we can contribute positively to the field while mitigating unintended consequences.

X. Project Completion Status

List of Features Implemented

Through constant debugging during setup of Phenix and Minimega tools, these solutions were found:

- Using Docker Volumes to resolve the mount binding error
- Changing the port that Phenix was running on since the default of 3000 was already in use on Windows machines
- Provided detailed documentation on common errors and the commands that resolved them or in the case that they were not resolved, the commands and solutions tested
- Documentation on how to set up the tools in different types of environments such as using docker containers for Windows Subsystem for Linux (WSL), docker containers for Kali Linux, and on building from source
- Experiments in Phenix were able to work, however, still some errors with Minimega that need to be ironed out

List of Features Not Implemented

Due to time constraints and ongoing setup challenges, we were unable to implement several key features of the Phenix tool, including:

- **Running Experiments with Phenix**: The ability to conduct controlled experiments within the SCADA environment.
- **Integration of the OT-SIM Protocol**: Seamless incorporation of the OT-SIM protocol into the virtual machines for enhanced operational testing.
- **Implementing Hardware in the Loop (HIL)**: Establishing connections with PLC devices for real-time control and monitoring.
- **YAML Configuration File**: A streamlined YAML file to automate the setup process and enhance user experience.
- **Simulated Cyber Attacks**: Utilizing a Kali Linux machine to test Phenix's resilience against cyber threats.

Performance Testing Results

Performance testing was significantly impacted by the challenges encountered during the setup process. Key issues included outdated dependencies within the Phenix tool, mount binding configurations that were not set as shared mounts, and permission conflicts between Phenix and Minimega. Consequently, our performance testing shifted focus to troubleshooting these specific issues across various environments, rather than achieving a stable operational benchmark.

Summary of Testing

Testing was conducted across several environments, each revealing challenges:

- **WSL (Windows Subsystem for Linux)**: Encountered port conflicts, mount binding not being shared, and potential permission errors that hindered setup.
- **Virtual Machine**: Faced persistent permission issues and complications with shared mount configurations.
- **Dual Boot Kali Linux**: Similar challenges with shared mounts and ongoing permission errors were noted.
- **Steam Deck**: Encountered difficulties related to shared mounts and permission mismanagement, preventing successful setup.

Each setup attempt resulted in errors, ultimately obstructing the establishment of functional Phenix instances.

Results of Usability Tests

Usability tests were conducted in each environment, yielding the following insights:

- **Mount Binding Issues**: The occurrence of mount binding problems—specifically, mounts not being configured as shared—was a common issue across multiple environments.
- **Permission Mismanagement**: Attempts to resolve mount binding issues by setting them as shared mounts often led to new errors, particularly concerning permission conflicts between Phenix and Minimega.

These usability tests underscored the complexity of the setup process and highlighted the need for a more robust configuration strategy.

XI. Future Work

The challenges encountered during the setup of Phenix have highlighted several areas for future improvement and development. Two primary avenues for future work include:

1. **Resolving Binding Mount Configuration**: One significant issue was the binding mount not being configured as a shared mount. While we were able to resolve this using docker mounts, future efforts could focus on investigating the underlying causes of this issue. This may involve:
    - Analyzing version histories and changelogs of Phenix to identify any modifications that could have impacted mount configurations.
    - Collaborating with the Phenix development community to determine best practices for configuring shared mounts in various deployment environments.
    - Conducting tests across different setups to identify reliable methods for achieving the desired shared mount behavior.
2. **Enhancing Permission Management between Phenix and Minimega**: Another critical area for future work is the permission management issues that arose when integrating Phenix with Minimega. These conflicts were particularly evident when attempting to implement solutions in the YAML Docker Compose file that utilized volumes to address shared mount concerns. To tackle this challenge, future work could include:
    - Developing a detailed configuration guide that outlines the necessary permission settings within the Docker Compose file, ensuring that Phenix and Minimega can operate cohesively.
    - Testing various configurations and settings in the Docker Compose file to identify effective solutions that prevent permission conflicts while maintaining functionality.
    - Engaging with the community to share findings and solicit feedback on configuration strategies that others have successfully employed.

Addressing these key issues allows for the creation of a more stable and user-friendly setup for Phenix, ultimately enhancing its utility as a SCADA tool for managing virtual machines in a virtual network.

XII. Lessons Learned

Throughout the project, we gained valuable insights into several critical topics related to setting up Phenix as a SCADA tool. Each lesson emerged from hands-on experience, particularly through debugging issues and exploring alternative solutions. Here's a detailed breakdown of what we learned:

1. **Docker Compose Files**: Our experience with Docker Compose files was crucial for orchestrating the various components of Phenix and Minimega. Initially, we struggled to configure the YAML file correctly, leading to issues such as service conflicts and misconfigured environment variables. Through debugging, we learned the importance of clearly defining each service's dependencies and network configurations. We also discovered that meticulous attention to indentation and syntax in YAML is essential to avoid deployment failures. This experience emphasized the need for precise documentation within our configuration files.
2. **Docker Volumes**: The transition to using Docker volumes was a pivotal moment in our project. After researching Docker volumes, we understood that they provide a more robust solution for data persistence and inter-container communication. This approach not only simplified our setup but also enhanced performance by avoiding the complexities of bind mounts. Our learning process involved testing various volume configurations and understanding their impact on file sharing and permission management between containers. This resolved mount binding issues, but the shared mount configuration led to permission conflicts.
3. **Setting Up Tools in Multiple Environments**: The necessity of testing Phenix in diverse environments—such as WSL, Virtual Machines, Dual Boot Kali Linux, and Steam Deck—highlighted the challenges across multiple enviroments. Each environment presented unique issues, such as port conflicts and permission errors. Through these experiences, we learned the importance of establishing a standardized testing protocol to streamline future setup processes. Documenting the specific requirements and known issues for each environment became a vital part of our workflow, ultimately helping to minimize setup time and errors.
4. **Mounts and Binding Mounts**: Our exploration of mounts and binding mounts was essential for understanding how Docker interacts with the host system. Initially, we faced significant challenges with shared mount configurations, which were important for Phenix's functionality. As we delved into the differences between various mount types, we learned that binding mounts can complicate permission management when not configured correctly. This knowledge allowed us to better troubleshoot and adjust our setup, reinforcing the idea that a deep understanding of Docker's mounting mechanisms is vital for effective container management.
5. **Docker Containers**: Working with Docker containers taught us the importance of isolation and reproducibility in software development. We learned how containers encapsulate dependencies, allowing for more predictable deployments. However, we also encountered issues when containers needed to interact with each other, especially regarding shared resources. Through debugging inter-container communication, we

recognized the value of defining clear networking rules and shared volumes in our Docker Compose configurations. This understanding significantly improved our ability to troubleshoot issues and optimize our container setups.

6. **Volume Permission Management**: Managing permissions for Docker volumes proved to be one of the more complex challenges we faced. The integration of Phenix and Minimega highlighted how permission mismanagement could lead to service failures. Our debugging efforts included giving full permissions within the Docker containers to allow proper access to shared volumes but to no avail. This experience underscored the need for careful planning around volume management in future projects.

To summarize, each lesson learned through this project not only enhanced our understanding of Docker and its ecosystem but also informed our approach to future software deployments. These insights will be instrumental in improving our workflows and ensuring smoother setups in subsequent projects.

XIII. Acknowledgments

A special thank you to our client, Nicolas Blair, for his support and collaboration throughout this project. Your insights and encouragement were instrumental in guiding our efforts. We also wish to express our gratitude to our advisor, Caleb Bartel, whose valuable guidance and support helped us navigate the challenges we encountered. Your contributions made a significant impact on our journey, and we sincerely appreciate all you have done to help us succeed.

XIIII. Team Profile

Steven Dillard

Bio: Senior majoring in Computer Science

Roles: Advisor point of contact and project report scribe

Colin Myers

Bio: Junior majoring in Computer Science

Roles: Client point of contact and scrum board overseer

Jackson Pow

Bio: Senior majoring in Computer Science

Roles: Presentation specialist

Appendix A – Key Terms

Term	Definition
Mini Mega	Minimega is a network emulation tool that allows users to simulate complex network topologies for testing various configurations and interactions in a controlled environment.
Phenix	Phenix is a SCADA tool designed for managing virtual machines within a virtual network, providing real-time monitoring and control capabilities for operational processes.
YAML File	The YAML file serves as a configuration document for Docker Compose, defining the services, networks, and volumes required for deploying Phenix and Minimega within Docker containers.