System For Pratical Evaluations of Network Administration Course

Diogo Nunes

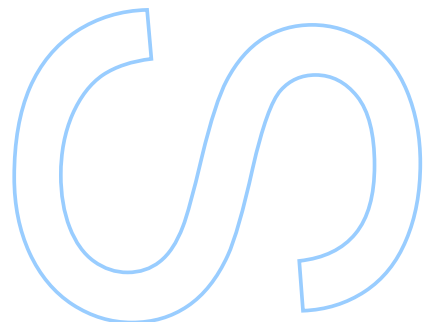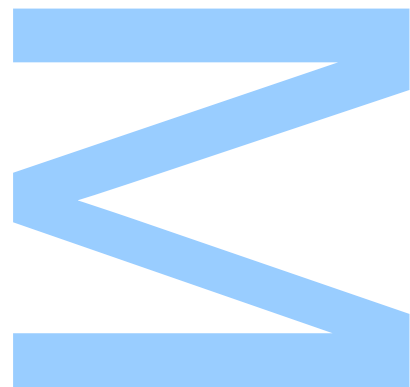Mestrado em Engenharia de Redes e Sistemas Informáticos
Departamento de Ciência de Computadores
2025

Orientador
Prof. Rui Prior, Faculdade de Ciências

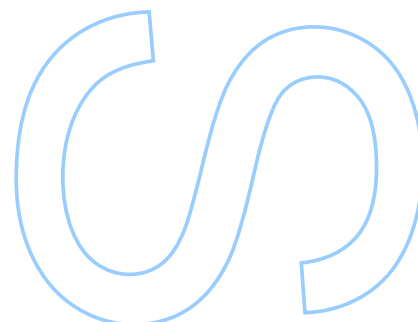Todas as correções determinadas
pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, _____/_____/_____

Acknowledgements

Acknowledge ALL the people!

Resumo

Este tese é sobre alguma coisa

Palavras-chave: física (keywords em português)

Abstract

This thesis is about something, I guess.

Keywords: Computer Science

Table of Contents

List of Figures

List of Abbreviations

API Application Programming Interface. 6, 8, 17

ASGI Asynchronous Server Gateway Interface. 11, 13– 16

CS Computer Science. 1

DCC Department of Computer Science. 4

GNS3 Graphical Network Simulator-3. 6, 14, 16– 18

HTTP Hypertext Transfer Protocol. 6, 10, 14, 16– 18

IOS Internetworking Operating System. 7

IOU IOS on Unix. 7

JSON JavaScript Object Notation. 6, 15, 17

KVM Kernel-based Virtual Machine. iv, 9

LXC Linux Containers. iv, 9

OAS OpenAPI Specification. 15

Proxmox VE Proxmox Virtual Environment. 8, 9, 14, 16– 18

QEMU Quick Emulator. 7, 9

REST Representational State Transfer. 6, 8

VM Virtual Machine. 8, 9

VPCS Virtual PC Simulator. 7

WSGI Web Server Gateway Interface. 10, 11, 14

1. Introduction

This chapter contextualizes challenges, outlining the limitations of current pedagogical tools and processes while framing the necessity for an automated, scalable solution. By dissecting the shortcomings of existing platforms to manual assessment burdens—we lay the groundwork for a system capable of network administration education.

1.1. Problem Statement

The digital transformation sweeping across industries has created unprecedented demand for skilled computer science professionals. While all Computer Science (CS) specializations face growing needs, network administration remains a foundational requirement. This persistent demand reflects networks' critical role as infrastructure supporting all digital systems.

Modern organizations require professionals who can design, configure, and troubleshoot increasingly complex network environments. Effective education must therefore bridge theoretical knowledge with practical implementation, particularly through hands-on evaluations that simulate real-world scenarios. Yet current assessment methods fail to meet these needs at scale, creating a growing gap between academic preparation and professional requirements.

1.1.1   Need for Automated Evaluation

Practical evaluations are essential to prepare students for real-world challenges, enabling them to apply theoretical knowledge and develop hands-on skills. However, available evaluation methods face significant limitations.

Creating a physical network environment for practical evaluations may be costly and challenging to scale for large student populations. While emulation and virtualization technologies offer cost-effective alternatives for creating flexible practice environments, they lack built-in automated assessment capabilities. Instructors must manually review each student's network topology configuration—a process that is:

- **Time-consuming**: Manual checks grow linearly with class size.

- **Error-prone**: Human reviewers may overlook misconfigurations.

- **Inconsistent**: Subjective grading criteria lead to unfair assessments.

Automating evaluations would:

- Reduce instructor workload, freeing time for student support.

- Ensure consistent, objective grading.

- Enable immediate feedback for learners.

1.1.2   Limitations of Current Solutions

Existing approaches to network education suffer from two critical gaps:

1. **Single-vendor focus**: Most tools (e.g., Cisco's packet tracer) are designed for specific vendor ecosystems, failing to prepare students for heterogeneous real-world networks where multi-vendor interoperability is essential.

2. **Missing evaluation component**: Our networking curriculum currently lacks practical assessments entirely, which means:

   - Students complete exercises without validation

   - No measurable feedback on configuration skills

   - Graduates enter industry with possible gaps in their knowledge

1.2.   Aims and Objectives

Building upon the foundational work of Santos [1] in automated network topology evaluation, this project has the following technical objectives:

(a) **Develop an automated evaluation environment**:

   - Create a system for assessing network administration exercises without manual intervention

   - Support multi-vendor device configurations (Cisco, Juniper, Linux-based)

   - Validate both connectivity and configuration compliance

(b) **Implement a cohesive back-end system**:

   - Transform prototype components from Santos [1] into a unified production system

- Develop capabilities to coordinate between:

  - Infrastructure provisioning

  - Network emulation

  - Evaluation automation

2. Background

The main focus of this chapter is to provide the reader with the necessary background to understand the context and technical foundations of this project. The goal of the system is to automatically evaluate network topologies by validating configurations and executing tests across various devices within a virtual network.

Achieving this requires the integration of multiple technologies, as the system must support a wide range of features to deliver a fully automated, end-to-end solution. Key concerns include not only functionality but also scalability, since multiple students may interact with the platform concurrently, each requiring an isolated working environment.

To support these requirements, this chapter introduces core concepts and tools such as virtualization, network automation, web frameworks and task processing. These components form the foundation upon which the system is built.

2.1. Overview of Used Technologies

2.1.1 Virtualization

Virtualization is the process of creating a virtual version of physical resources, such as routers, switches, or even entire computers. In the context of this project, it is used to create virtual machines to provide students with a work environment and virtual networks, comprised of various types of virtualized devices. This approach enhances scalability and reduces costs, as it allows multiple virtual machines to be run on a single physical machine.

Virtualization can be categorized into **emulation** and **simulation**.

- Emulation is the process of creating a virtual version of a physical device in software, replicating its behavior exactly—including any bugs and limitations. This is useful for various things like testing software on different platforms, running legacy software on modern hardware and even running potentially harmful software in a safe isolated environment. Emulation will be used to provide students with a work environment to test their network configurations, as well as to emulate certain network devices.

- Simulation models the behaviour of a device, without replicating the underlying hardware or software. This results in a simpler less resource intensive

model, though it may not fully capture the real device's behavior. Simulation will be used to simulate the behaviour of certain, simpler and generic, network devices.

2.1.2 GNS3

Graphical Network Simulator-3 (GNS3) is an open-source graphical network emulator software that allows the user to create complex network topologies and interact with the various devices in it. It is widely used for educational purposes and is often used in preparation for professional network certifications like the Cisco Certified Network Associate (CCNA).

GNS3 employs a simple drag and drop interface to allow users to add new devices, make links between them and even add textual annotations. The software allows users to interact with the devices by way of a console or even a GUI if the device supports it. The software also allows users to export their topologies to be shared with others, which can be useful for teachers to provide students with a pre-configured topology to work on.

Additionally, the software supports packet capturing which is essential for students to develop their debugging and troubleshoting skills. Finally it can also be interacted with via a Representational State Transfer (REST) Application Programming Interface (API) which is of particular interest for this project.

2.1.3 Architecture

The software can be employed in a variety of ways due to its architecture [2] that separates the user interfaces that it offers, namely the locally installed gns3-gui as well as the browser accessible gns3-web, from the gns3-server that runs the emulations and the controller who orchestrates everything.

2.1.3.1 Controller

The controller is integrated in the gns3-server project and is responsible for communicating with all the other components of the software. The controller is a singleton, meaning there should only be one instance of it running at any given time, and it does not support concurrent requests. It is able to control multiple compute instances if so desired, each capable of hosting one or more emulator instances, varying depending on their complexity. The controller also exposes the REST API allowing

the ability to interact with the software programatically. All communication is done over Hypertext Transfer Protocol (HTTP) in JavaScript Object Notation (JSON) format and there is support for basic HTTP authentication as well as notifications via websockets.

2.1.3.2 Compute

The compute is also integrated in the gns3-server project and controls the various emulators required to run the nodes in the topology. The list of currently supported emulators is:

- **Dynamips** - Used to emulate Cisco routers and basic switching.

- **IOS on Unix (IOU)** - Used to emulate Cisco Internetworking Operating System (IOS) devices.

- **Quick Emulator (QEMU)** - Used to emulate a wide variety of devices.

- **Virtual PC Simulator (VPCS)** - A basic program meant to simulate a basic PC.

- **VMware/VirtualBox** - Used to run virtual machines with nested virtualization support.

- **Docker** - Used to run docker containers.

2.1.3.3 GUI

The GUI is composed of two separate but with mostly identical functionality, namely the gns3-gui and the gns3-web projects. The gns3-gui project is a desktop application that is used to to interact with a local or remote gns3-server instance. It is written in Python and uses the Qt framework for the graphical interface. The gns3-web is a web application that is accessed via a web browser it is still in a beta stage but is already capable enough to be used as a substitute for the gns3-gui.

2.1.4 Linux

Linux is a core component of this project, as it is the kernel of all operating systems used to host all the services provided.

- **Containerized Web Application** - The web application runs inside an LXC container on Proxmox. Since LXC containers share the host kernel, the containerized web application benefits from efficient resource usage while maintaining isolation from the host system.
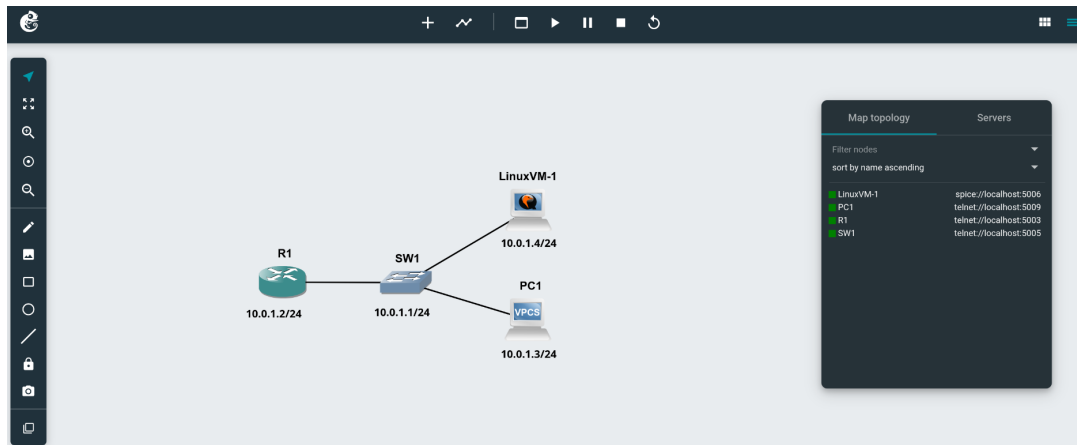
Figure 2.1: A simple network topology example in the GNS3 Web UI

- **Virtual Machines hosting GNS3** Proxmox also hosts virtual machines run-ning Linux-based Ubuntu GNS3 servers. These VMs provide students with isolated environments to configure and test network topologies, benefiting from the flexibility of full virtualization through KVM, giving them the ability to virtu-alize any type of network device. Students only interact with these Virtual Machine (VM)s via the GNS3 web Interface. Students interact with these VMs using a browser, accessing the gns3-server instance running on them. Thus the students never have to interact with the underlying operating system, and so these machines do not require a desktop environment, which helps freeing up resources leading to better scalability.

- **Proxmox Virtual Environment (Proxmox VE)** - Proxmox is a Linux-based open-source platform for enterprise-level virtualization. It is based on the De-bian Linux distribution.

2.1.5 Proxmox Virtual Environment

Proxmox VE is an open-source platform designed for enterprise-level virtualiza-tion [3]. It is based on the Debian distribution of Linux and provides a web-based interface for managing virtual machines and containers. It is widely used in data centers and cloud environments, as it provides a scalable and reliable solution for virtualization.

Proxmox VE bundles several core services that can be interacted with via shell commands, a web interface or even by using the Proxmox VE REST API. These allow the user to interact with every service provided by Proxmox VE, in a plethora of ways, depending on the user's needs, skills and preferences. The web interface

is the most user-friendly way to interact with the platform, as it provides a graphical interface for managing the cluster. The shell commands provide a more direct way to interact with the platform, allowing for more complex operations to be performed and opening the doors to scripting and automation. Finally, the Proxmox VE REST API allows for programmatic interaction with the platform, enabling users to create custom applications that can interact with the platform.

2.1.5.1 Virtualization Technologies

Proxmox VE supports the the deployment and management of two distinct types of virtualization, namely, Kernel-based Virtual Machine (KVM)- based VMs and Linux Containers (LXC)-based containers.

Users can interact with these virtualized environments via NoVNC, a simple web-based VNC client or SPICE which is a more feature-rich protocol that provides better performance and more features than VNC. Both of these protocols support the use of a console-based interface, aswell as a full desktop graphical interface.

2.1.5.2 KVM

KVM is a virtualization solution provided by the Linux kernel. It leverages the hardware virtualization extensions of modern processors to provide a full virtualization experience at near-native speeds. Supports a wide range of guest operating systems making it a good choice for general purpose virtualization.

In Proxmox VE, KVM is used as the core component for running virtual machines and is used alongside QEMU.

2.1.5.3 LXC

Containerization is an operating system-level virtualization method that packages an application and its dependencies together into an isolated environment. Contrary to tradional VMs, containers dont emulate hardware or require a guest operating system relying instead on the host's kernel. This approach leads to a faster and more lightweight virtualization solution, as they consume less memory and cpu resources.

LXC creates full system containers, capable of simulating a complete Linux distribution providing users with an environment that behaves like a traditional VM but with the speed and efficiency of a container. LXC start much faster than VMs making them ideal for scenarios requiring rapid deployment and/or scaling.

However, it's important to note that while containers offer a degree of isolation, they do not provide the same level of security as VMs. This means that while they may not always be a suitable replacement for VMs.

2.1.6 LDAP

Lightweight Directory Access Protocol (LDAP) is the foundation of user and device management in many institutions. Universities can rely on LDAP and Microsoft's Active Directory (its enterprise implementation) to handle student, faculty accountsa and lab computer access.

One of LDAP's most popular implementations, OpenLDAP, had its initial release in 1998. The protocol's longevity stems from its efficiency at handling large-scale authentication so much so that despite newer alternatives existing, LDAP remains entrenched in academic environments due to its reliability and universal adoption. For our project, LDAP integration enables students access to the system using their existing university credentials.

Two key factors make LDAP particularly valuable for this project: its standardized approach to user management and pre-existing deployment in our target educational environments. Given the extensive use, it's logical for our system to interact with LDAP in order to correctly authenticate users.

2.2. Virtualized Lab Environments

The combined use of Proxmox VE as a virtualization platform and GNS3 for network emulation presents a cost-effective solution for scalable networking education. This approach offers significant benefits over physical lab infrastructures:

- **Resource Efficiency**: Single physical host can support multiple concurrent student environments

- **Operational Characteristics**:
  - Accelerated environment provisioning through templates
  - State preservation via Proxmox VE's snapshot/restore functionality
  - Support for diverse network operating systems through virtualization technologies

2.2.1 Implementation Trade-offs

Virtualized environments introduce specific technical considerations:

- **Fidelity**: Variable emulation fidelity depending on emulator maturity

- **Containerization Limitations**:
  - Restricted access to nested virtualization features
  - Challenges in device passthrough configurations

2.3. Python Web Frameworks for API-Based Systems

2.3.1 Python

Python is a high-level, interpreted programming language renowned for its readability and versatility. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming, making it suitable for a wide array of applications. In the context of this project, Python serves as the primary programming language as its extensive standard library and supportive community contribute to efficient development and maintenance of the project's codebase.

2.3.2 WSGI

The Web Server Gateway Interface (WSGI) is a pivotal standard for Python web application deployment, defining a consistent interface between web servers and Python web applications/frameworks.

Prior to WSGI's introduction, Python web frameworks were typically written against various server-specific APIs such as CGI, FastCGI, or mod_python. This diversity led to compatibility issues, limiting developers' choices of web servers and frameworks, as not all frameworks supported all web servers and vice-versa. To address this fragmentation, WSGI was created as a standardized interface, promoting portability and flexibility in deploying Python web applications.

WSGI serves as a bridge, enabling web servers to communicate with Python applications. It specifies a simple and universal interface for web servers to forward requests to Python applications and for those applications to return responses. This standardization allows developers to choose from a variety of web servers and Python frameworks without compatibility concerns[4].

Introduced in 2003 as PEP 333, WSGI was later updated to PEP 3333 in 2010 to accommodate Python 3. These specifications outline how web servers and Python

applications should interact, ensuring a consistent and reliable deployment environment across different platforms.

The WSGI standard consists of two main components:

- **Server/Gateway Side** - Responsible for receiving HTTP requests from clients and passing them to the Python application. Then receives the response from the application and forwards it to the client.

- **Application** - The Python application that processes requests and returns responses.

Additionally WSGI has support for middleware components. WSGI middleware is a Python callable that wraps another WSGI application to observe or modify its behavior. Middleware can perform various functions, including request preprocessing, response postprocessing, session management, and security checks. This modularity allows developers to add functionality to their applications in a reusable and maintainable manner.

The separation defined by WSGI allows for flexibility and scalability in deploying Python web applications.

Python WSGI applications often use built-in servers, during development, provided by frameworks like Flask. However, these servers typically aren't fully featured and aren't suitable for production environments. In production, WSGI servers act as intermediaries between web servers (e.g., NGINX or Apache) and Python applications, handling incoming requests and serving responses efficiently.

2.3.2.1 Flask

Flask is a web application micro framework written in Python, adhering to the WSGI standard, designed to facilitate the development of web applications by providing essential tools and features. Classified as a microframework, Flask does not require particular tools or libraries, instead choosing to focus on simplicity and extensibility[5].

An example of how easy it is to develop a basic web application with flask is provided in the following small piece of code.

2.3.3 ASGI

Asynchronous Server Gateway Interface (ASGI) is an interface specification for Python web servers and applications. It is considered a spiritual successor to WSGI,

---

Algorithm 1 Flask Hello World

```
1: from flask import Flask
2: app = Flask(__name__)
3:
4: @app.route('/')
5: def hello_world():
6:     return 'Hello, World!'
7:
8: if __name__ == '__main__':
9:     app.run()
```

---

designed to provide a standard interface for asynchronous communication. ASGI was developed to address the limitations of WSGI, which was primarily designed for synchronous applications. Unlike WSGI, ASGI supports handling multiple requests concurrently, making it suitable for modern web applications that require real-time features such as WebSockets, long-lived connections, background tasks or the use of Python's async features.

As development progressed, asynchronous task handling became a more central requirement, initially addressed by integrating task queues. However, due to resource overhead and deployment complexity, they were phased out. This shift prompted an evaluation of frameworks that offered native support for asynchronous operations.

2.3.3.1 FastAPI

FastAPI is a modern, high-performance web framework adopting the ASGI standard. It leverages open standards, such as OpenAPI Specification (OAS), for defining path operations, parameters, and more, which in turn is based on the JSON schema. FastAPI relies entirely on Python type declarations, making it more intuitive and lowering the barrier to entry to new developers. This approach also simplifies the understanding and maintenance of the codebase.

Built on top of Starlette, a lightweight ASGI framework, and Pydantic, a data validation library. FastAPI combines the strengths of both to provide a powerful and flexible framework for building APIs with automatic data validation, serialization and documentation generation, all of which significantly enhance developer productivity.

Another key feature of FastAPI, being ASGI-compliant, is its built-in support for asynchronous programming, allowing developers to write non-blocking code using

Python's *async* and *await* keywords. This is particularly useful for I/O-bound operations, such as database queries or network requests, as it allows the application to handle multiple requests concurrently without blocking the application which is essential in projects such as this one where multiple concurrent HTTP calls are made to interact with multiple devices and services concurrently, such as GNS3 and Proxmox VE.

Another powerful feature of FastAPI is its dependency injection system, that is very easy to use as it is automatically handled by the framework itselft. This allows for a clean and modular codebase, as dependencies can be easily injected into the various components of the application. This is especially useful in larger applications, where managing dependencies can become complex and cumbersome.

This change laid the groundwork for more efficient handling of I/O-bound operations—such as network interactions with Proxmox or GNS3, which will be of importance in future iterations of the project while also streamlining endpoint development thanks to FastAPI's built-in request parsing, background task support, and integrated dependency injection system.

2.4. Long running task processing approaches

2.4.1 The need for asynchronous processing in API-heavy applications.

Modern API-driven applications benefit tremendously from asynchronous programming paradigms to handle concurrent operations efficiently. Traditional synchronous execution models, where each request blocks thread execution until completion, prove inadequate for systems requiring high throughput and responsiveness. This limitation becomes particularly apparent in projects like ours, which relies heavily on HTTP calls to various devices and services.

The asynchronous model, implemented through Python's `async/await` syntax, offers several critical advantages:

- **Improved Resource Utilization**: A single thread can manage multiple concurrent I/O operations by yielding control during waiting periods

- **Enhanced Scalability**: Systems can handle more concurrent users with fewer hardware resources

- **Responsive Performance**: The application remains reactive even during long-running operations

Although Python has native asynchronous capabilities, libraries must be written with these in mind, meaning some may have limited or even no support for these capabilities.

2.4.2 Asyncio

Asyncio is Python's built-in library for writing concurrent asynchronous code. It serves as the foundation for asynchronous operations in many Python frameworks, enabling high-performance networking, web servers, database connections, distributed task queues, etc..

Asynchronous I/O can be useful in cases of time-consuming operations, as while awaiting the finish of said tasks, it relinquishes control so that other code can run in the meantime. This approach is particularly well-suited for I/O-bound operations—such as network communication, file access, or database queries—where tasks would otherwise spend a significant amount of time waiting for external operations that are outside of our control to complete. Rather than blocking the entire application during such waits, Asyncio allows other tasks to execute in the meantime, leading to more efficient resource utilization and improved throughput.

Overall, Asyncio provides the concurrency model that underpins efficient I/O performance. By embracing this model, the project benefits from improved responsiveness, lower latency, and better scalability—especially under workloads that involve heavy interaction with external services.

Frameworks that leverage these capabilities natively, provide the foundation for building responsive, scalable API services.

2.4.3 Celery

Celery is an open source distributed task queue focused on real-time processing but also offers support for task scheduling. It is implemented in Python, but the underlying protocol can be implemented in any language. Celery requires a message broker to function, such as Redis or RabbitMQ, which are responsible for queuing and distributing tasks from producers (clients) to consumers (workers).

When integrating Celery into their projects developers must mark functions they want to be processed as tasks with Celery provided decorators (e.g. `@app.task`) which allows workers to then execute them asynchronously. The system shines

in projects requiring heavy computational or schedules jobs, but brings with it non-negligible operational, developmental and resouce overhead, doubly so if the project didn't already include the use of message brokers.

2.5. System administration automation tools

Modern system administration increasingly relies on automation tools to manage complex infrastructure while maintaining reliability and reproducibility. In our context, configuration management systems serve as the foundational layer for ensuring well-configured device states across network environments.

2.5.1 Ansible

Ansible is a widely adopted open-source automation platform that simplifies configuration management, application deployment, and task automation through a declarative YAML-based approach. Unlike imperative scripting solutions, Ansible employs playbooks to define system states, making automation accessible to both developers and operations teams while maintaining robust capabilities for complex workflows.

The platform operates on an agentless architecture, utilizing Secure Shell (SSH) for connectivity, which eliminates the need for persistent software on managed nodes. This design choice significantly reduces deployment overhead while maintaining secure communication through standard protocols. Ansible's push-based execution model allows for immediate task execution across entire device inventories without requiring pre-installed clients.

Ansible remains a valuable tool in task automation and orchestration but, as was already discussed in[1] there are several barriers to the adoption of Ansible in this project, mainly the difficulties encountered by utilizing the Telnet protocol for communications with network devices that dont support the SSH protocol .

2.5.2 Nornir

Nornir is an open-source automation framework written in Python, designed to provide a flexible and efficient approach to network automation tasks[6]. Unlike other automation tools that utilize customized pseudo-languages, Nornir leverages pure Python code, offering developers the full power and versatility of the Python ecosystem.

Nornir supports multi-threaded task execution, allowing operations to run parallel across multiple devices. This capability enhances efficiency and reduces the time required enabling easy scaling to a large number of devices.

The framework provides a robust inventory management system, enabling the organization of devices into groups and the assignment of specific tasks to these groups. This structure facilitates targeted automation and simplifies complex network operations.

Finally, thanks to Nornir's architecture, it is highly extensible through its plugin system, allowing users to create custom plugins for inventory management, task execution, and result processing. This modularity ensures that Nornir can adapt to a wide range of network automation scenarios.

Nornir is particularly well-suited for tasks such as configuration management and state validation which makes it highly desirable in the context of this project. Its ability to handle concurrent operations will also ensure it can scale alongside the rest of the project.

3.   Related Work

This chapter focuses on placing the current project within the context of existing solutions and related work. The primary goal of this project is to develop a system capable of automatically evaluating network topologies by validating device configurations and executing tests across a virtual network.

While automated assessment systems are well established in the field of programming education—receiving student-submitted code and running it against predefined test cases—equivalent systems for network exercises are far less common. Tools like Mooshak and similar platforms have proven effective for evaluating programming assignments and are widely adopted in academic settings.

At first glance, adapting these approaches to network topologies might seem straightforward. However, network evaluation introduces unique challenges such as the need for per-student virtual environments, real-time communication with multiple devices, and stateful, distributed configurations. This chapter explores existing tools like Mooshak and Packet Tracer, highlighting their capabilities, limitations, and how this project builds upon or diverges from them.

3.1.   Programming Evaluation Systems

While not directly related, they are the main inspiration for this project. Programming evaluation systems are widely deployed in universities and other educational institutions. These systems receive as input code from students and subsequently run tests on it, outputting a score and even being configurable to provide students the first test case that they failed in, guiding students to the solution without handing it out.

The main differentiator between these systems and the one proposed in this project is the ability to solve a network exercise using multiple configurations across multiple devices, while programming evaluation systems will expect the same output every time, given the same input.

Another key difference is the fact that programming evaluation systems dont always provide a working environment for the students to test their code, owing to the fact that students might prefer to user their own development environment for initial development and testing. This project aims to provide a working environment for

students, as setting up a networking lab can be a daunting task for students, especially when they are just starting out. By providing a pre-configured environment, students can focus on learning the concepts and skills they need to succeed in their studies, rather than spending time troubleshooting their setup.

3.1.1 Mooshak and lessons learned

In our context, in the Department of Computer Science (DCC), Mooshak and Codex are commonly deployed to be used in the context of classes and even exams and programming contests.

Mooshak is a web-based system for managing programming contests and also to act as an automatic judge of programming contests [9]. It supports a variety of programming languages like Java, C, etc. Under each contest students will find one more problem definitions each containing varying sets of test cases in input-output pairs. After submiting their solution, the system will compile and run the code against the test cases giving a score based on the the amount of test cases passed.

Mooshak provides a structured approach to test coding and problem solving skills. It begins by offering a problem statement coupled with an optional image and an example test case, normally in the form of input and expected output. Users can submit their proposed solution by uploading a file with their code. The system then evaluates the provided solution against multiple pre-defined test cases, validating the output against the know-good output, and giving feedback in the form of a score based on the number of test cases passed. The system may also be configured to have time and/or memory constraints, to ensure that temporal and spatial complexity are also taken into account.

All of these, serve to provide a thorough evaluation of the student's solution, which can help guide a student to better their coding and problem solving skills.

The system can also differentiate between differing types of errors, such as not giving the expected output, poorly formatted output, failure to compile or even exceeding the time limits. Mooshak also includes some features designed to drive competition between students, like a real time leaderboard and the ability to have more than 100% of the score for a given contest.

The system however is not without its limitations as it uses plain text files for its test cases and validates the output of student's code character by character, which can lead to false negatives if the output is not formatted exactly as expected.

3.2. Cisco Packet Tracer

Cisco Packet Tracer is a network **simulation** tool developed by Cisco Systems, widely used in academic environments to teach networking concepts and prepare students for certifications such as the Cisco Certified Network Associate (CCNA). It offers a visual interface for building and simulating virtual network topologies using a variety of Cisco devices, including routers, switches, and end devices.

While Packet Tracer is highly accessible and effective for introducing networking fundamentals, it is a closed-source, proprietary tool limited to simulating Cisco hardware and IOS features. Its functionality is optimized for teaching purposes rather than for flexibility, extensibility, or integration into larger automated workflows.

In contrast, this project aim to allow for a more realistic and extensible lab environment. The use of real operating systems and support of a wide range of vendor platforms for routers and switches, aswell as Linux-based virtual machines is highly desirable. This allows for a more realistic experience, as students will be able to work with the same tools and operating systems that they will encounter in real-world scenarios.

Therefore, while Cisco Packet Tracer remains a valuable educational tool, the needs of this project called for a more flexible and open architecture, which is better addressed by other tools.

4.   System Architecture & Design

Delivering reliable, scalable automated assessment of virtualized networks requires a system built on solid principles of virtualization and automation. This chapter outlines the architecture of the proposed system, detailing the key components and how they interact to enable seamless evaluation of student-submitted network exercises.

The system is designed to provide each student with a working environment where custom network topologies can be deployed, configured, and tested. To achieve this, the platform integrates several technologies—such as GNS3 for network emulation, Proxmox VE for virtualization, and Nornir for configuration testing—alongside an asynchronous web-based API layer for user interaction and system communications.

This section provides a high-level overview of the system, the rationale behind its design choices, and the fundamental components that make up its architecture.

4.1.   System Architecture Overview

The system architecture is designed to provide a robust and scalable solution for the automated assessment of network topologies. The architecture is divided into several key components, each responsible for a specific aspect of the system's functionality. The main components of the system architecture are as follows:

- **Web App:** The web app serves as the main interface for users to interact with the system. It provides endpoints for evaluation, creation and viewing available exercises. The app is designed to be asynchronous where possible, allowing for efficient handling of multiple requests simultaneously.

- **Proxmox VE:** Proxmox VE is responsible for creating and managing VMs that host the network devices used in the exercises. This layer interacts with the web app to create and manage the VMs based on the creation of new exercises or students. All communication with Proxmox VE is done asynchronously through the Proxmox REST API, which allows for efficient communication, keeping the web app responsive, while also keeping the components decoupled.

- **GNS3:** GNS3 is used to emulate all the components of the virtual networks to be configured by students, using various types of virtualization detailed earlier.

Communication with GNS3 is done through the GNS3 REST API by the
web app, due to a lack of robustness of the GNS3 API.

[synchronously (will it stay this way)]

- **Nornir:** This automation framework is used for validating device configurations. It connects to the virtualized devices, executes commands, and compares the output to expected results to determine correctness.

4.2. Component Breakdown

4.2.1 Web Application

The web application serves as the primary interface through which users interact with the system. It is built using the FastAPI framework and follows an asynchronous-first, modular architecture that scalable interactions with other system components.

The application exposes a REST API that supports endpoints for user authentication, exercise creation, virtual machine management, and configuration validation. It acts as the coordinator for the entire system, triggering operations in Proxmox VE, Proxmox VE, and Nornir based on user actions.

Wherever possible, asynchronous I/O is employed to prevent blocking during operations such as API calls to Proxmox. Multiprocessing is also utilized to handle configuration validation. This keeps the system responsive and performant, especially when handling multiple simultaneous requests from different users.

Internally, the application is designed to be stateless and maintain minimal runtime state. Most essential information—such as user accounts, defined exercises, and student-to-VM mappings—is persisted in a relational database rather than stored in memory. Configuration values such as API tokens, base URLs, and database credentials are injected via environment variables to decouple deployment-specific settings from the application code. This design improves reliability, supports concurrent usage, and enables horizontal scalability if deployed across multiple instances.

To ensure maintainability and modularity, interactions with external services like Proxmox VE and GNS3 are isolated in dedicated modules. These serve as abstraction layers between the application logic and third-party APIs, exposing clean, reusable interfaces while hiding low-level implementation details. For example, Proxmox-related operations such as VM creation and deletion are handled in a separate module (e.g services/proxmox.py), as are all GNS3-related tasks. This separation of concerns improves the structure of the codebase and simplifies future maintainability by being more readable.

To help with development and testing, the application automatically generates OpenAPI-compliant documentation, allowing developers to explore and interact with available endpoints. This self-documenting behavior streamlines integration testing and encourages a more agile development process.

Finally, to safeguard user data and infrastructure control points, the application enforces secure authentication mechanisms using JSON Web Token (JWT) ensuring that only authorized users can trigger actions on shared resources.

JWT doesnt quite work yet

4.2.2 Proxmox Virtual Environment

Proxmox VE functions as the virtualization backbone of the system, enabling the creation and management of Linux-based VMs used in the automated assessment workflow. Each VM runs a lightweight Linux operating system with a dedicated GNS3-server instance, providing a self-contained environment for deploying and configuring virtual networks and their components.

The lifecycle of a VM in the system begins when a new exercise is created. At this point, the platform clones a pre-configured base template VM available in Proxmox VE. This VM is then started and automatically configured: the provided GNS3 project file is imported, and a sequence of user-defined commands is executed within the GNS3 environment. Once the setup is finalized, the configured VM is converted into a new template VM that is tailored to that exercise.

From this template, a "work" VM is created for every active student. These VMs are exact clones of the configured template and serve as the student's personal lab environment for a given exercise. Each student thus receives a consistent starting point for the exercise while working within an private instance of the network topology.

By default, these VMs are not strictly isolated from each other at the network level. However, isolation can be enforced by dynamically applying firewall rules that restrict each student VM's network access to a single designated validation host—the same host that also runs the web application at the current stage of development. This approach ensures a secure and controlled assessment environment while retaining the flexibility to scale or adjust network policies as needed.

The Proxmox VE infrastructure is utilized with scalability in mind. The use of linked clones and storage-efficient backing filesystems, in this case LVM-thin, allows the system to rapidly provision VMs while minimizing storage usage. Load balancing

across multiple Proxmox VE nodes is also possible, although the current implementation assumes a single-node deployment for simplicity.

All Proxmox VE-related operations—such as cloning, starting, templating, and deletion—are fully automated and triggered by the web application. Under normal operation, no manual intervention using the Proxmox VE web UI or shell utilities is required; such intervention is only necessary when the system's error handling mechanisms detect failures that cannot be automatically resolved. To securely execute these operations, the application authenticates to the Proxmox VE API using token-based authentication. The required credentials and configuration parameters are securely injected via environment variables, while the time limited token is stored in memory, ensuring that only authorized and properly configured processes can interact with the Proxmox VE infrastructure.

4.2.3   GNS3 Network Emulator

4.2.4   Nornir Automation Framework

4.2.5   Storage and Data Model (Maybe)

5. Implementation

6. Testing & Evaluation

7.   Conclusion & Future Work

References

[1] P. M. C. Santos, "Environment for practical evaluations in network adminis-
tration," Master's thesis, Faculdade de Ciências da Universidade do Porto,
Porto, Portugal, 2024, accessed: March 21, 2025. [Online]. Available:
https://hdl.handle.net/10216/164899 [Cited on pages 2 and 15.]

[2] GNS3 Documentation, "Architecture," n.d., accessed: March 15, 2025. [Online].
Available: https://docs.gns3.com/docs/using-gns3/design/architecture/ [Cited
on page 5.]

[3] Proxmox Server Solutions GmbH, *Proxmox VE Administration Guide*, 2025,
accessed: March 21, 2025. [Online]. Available: https://pve.proxmox.com/pve-
docs/pve-admin-guide.html [Cited on page 7.]

[4] P. J. Eby, *PEP 333 – Python Web Server Gateway Interface v1.0*,
Online, 2003, accessed: March 21, 2025. [Online]. Available: https:
//peps.python.org/pep-0333/ [Cited on page 10.]

[5] P. Projects, "Flask documentation," Online, 2025, accessed: March 21,
2025. [Online]. Available: https://flask.palletsprojects.com/en/stable/ [Cited on
page 11.]

[6] D. Barroso, "Nornir: The python automation framework," Online, 2025,
accessed: March 21, 2025. [Online]. Available: https://nornir.readthedocs.io/
en/latest/ [Cited on page 15.]

[7] K. Reitz and R. Contributors, "Requests: Http for humans," Online, 2025,
accessed: March 18, 2025. [Online]. Available: https://requests.readthedocs.
io/en/latest/

[8] T. Christie and E. Contributors, "Httpx: A next generation http client
for python," Online, 2025, accessed: April 04, 2025. [Online]. Available:
https://www.python-httpx.org/

[9] J. Leal and F. Silva, "Mooshak: a web-based multi-site programming contest
system," *SOFTWARE-PRACTICE & EXPERIENCE*, vol. 33, no. 6, pp. 567–581,
MAY 2003. [Cited on page 18.]

Appendix Title Here

Write your Appendix content here.