UCLouvain

École polytechnique de Louvain

Apprentissage actif des réseaux informatiques par le débogage

Conception de laboratoires Kathará avec correction automatique sur INGInious

Auteur: **Noah Van Horenbeke**
Promoteur: **Olivier Bonaventure**
Lecteurs: **Anthony Doeraene, Ramin Sadre, Rémi Van Boxem**
Année académique 2025–2026
Master [120] en sciences informatiques

Remerciements

Je tiens à remercier avant tout mon promoteur, Olivier Bonaventure, pour m'avoir permis, à travers ce sujet, de contribuer à l'amélioration de l'apprentissage des futurs étudiants.

Je souhaite également exprimer ma gratitude à Anthony Doeraene, qui m'a particulièrement bien accompagné tout au long de ce travail. Il a toujours été disponible pour répondre à mes questions et m'a apporté des retours à la fois pertinents et constructifs.

Je remercie aussi Anthony Gego, qui m'a aidé dans la configuration d'INGInious pour mes exercices, ainsi que Rémi Van Boxem, avec qui j'ai partagé un intérêt commun pour l'amélioration de l'apprentissage via Kathará, et qui m'a également fourni des retours précieux.

Enfin, je remercie ma famille et mes amis pour leur soutien tout au long de la réalisation de ce mémoire, même si le domaine de l'informatique leur est parfois éloigné.

Résumé

L'enseignement des réseaux informatiques et de la cybersécurité requiert des exercices pratiques confrontant les étudiants à des environnements réalistes. Les outils d'émulation existants ne proposent cependant pas de mécanisme intégré pour distribuer des exercices personnalisés, corriger automatiquement les soumissions et fournir un feedback structuré à grande échelle.

Ce mémoire présente un framework open source comblant ce manque en combinant Kathará, un émulateur réseau basé sur des conteneurs, et INGInious, une plateforme de correction automatique. L'idée pédagogique centrale est de placer l'étudiant dans une démarche active de diagnostic : plutôt que de construire une configuration depuis zéro, chaque étudiant reçoit un réseau préconfiguré contenant des erreurs intentionnelles qu'il doit identifier et corriger. Afin de garantir l'intégrité académique, chaque étudiant se voit assigner une instance unique du laboratoire, appelée variant, présentant une combinaison d'erreurs et un schéma d'adressage distincts, générés automatiquement à partir d'un ensemble de templates paramétrés.

Plusieurs laboratoires ont été développés et mis en ligne, couvrant des thématiques telles que le routage statique, BGP, OSPF et les attaques réseau.

La sécurité et la scalabilité de l'infrastructure déployée sont analysées, et les résultats montrent que le mécanisme de scheduling d'INGInious, s'appuyant sur des agents dédiés chacun connecté à sa propre machine virtuelle Docker, résout efficacement les problèmes de concurrence identifiés lors des premiers tests. Le framework est conçu pour être extensible et réutilisable, constituant une base documentée sur laquelle d'autres enseignants peuvent s'appuyer pour enrichir leurs cours.

Table des matières

Chapitre 1

Introduction

1.1 Contexte et motivation

L'utilisation d'outils basés sur l'intelligence artificielle par les étudiants a considérablement évolué ces dernières années. Les modèles de langage de grande taille (Large Language Models, LLM), tels que les assistants conversationnels, sont aujourd'hui capables de fournir des explications détaillées sur de nombreux concepts théoriques et de résoudre une large variété de tâches académiques et techniques [1]. Cette évolution transforme progressivement les méthodes d'apprentissage et soulève de nouvelles questions pédagogiques, notamment dans les domaines techniques comme les réseaux informatiques et la cybersécurité.

Dans ce contexte, il devient essentiel de proposer des activités pédagogiques qui favorisent un apprentissage actif et immersif, permettant aux étudiants d'interagir directement avec les concepts étudiés plutôt que de se limiter à une approche purement théorique. En particulier, les exercices pratiques permettant de manipuler des systèmes réels ou simulés encouragent les étudiants à développer une compréhension plus profonde des concepts abordés. Les travaux pratiques constituent ainsi un élément central de l'enseignement en informatique, car ils permettent de confronter la théorie à des situations concrètes.

Ce projet de mémoire s'inscrit dans une réflexion menée autour de l'amélioration des travaux pratiques proposés dans les cours de réseaux informatiques. Plus particulièrement, la possibilité de proposer aux étudiants des environnements expérimentaux leur permettant de manipuler des configurations réseau concrètes constitue un enjeu important pour l'apprentissage des concepts liés aux réseaux et à la cybersécurité.

Dans ce contexte, l'utilisation d'environnements de laboratoires virtuels, permettant de reproduire des topologies, protocoles et configurations réseau dans un environnement isolé et reproductible, apparaît comme une solution pertinente pour

concevoir des exercices pratiques reproductibles et automatisables. Ces environnements permettent de reproduire des topologies réseau complexes tout en restant accessibles aux étudiants sur leurs propres machines.

C'est dans cette perspective que l'outil Kathará a été retenu comme base technologique pour ce projet. Kathará est un framework open-source permettant de créer et d'exécuter des laboratoires réseau virtuels en s'appuyant sur des conteneurs Docker. Chaque nœud de la topologie est représenté par un conteneur, ce qui permet de reproduire des environnements réseau réalistes tout en conservant une infrastructure légère et facilement déployable.

L'un des éléments particulièrement intéressants de cet outil est le composant *Kathara Lab Checker*, qui permet de mettre en place des mécanismes de correction automatique avancés pour les travaux pratiques. Ce système permet notamment de vérifier automatiquement la configuration des machines virtuelles et le comportement du réseau émulé [2]. Cette fonctionnalité ouvre la possibilité de proposer des exercices interactifs dans lesquels les étudiants doivent analyser et corriger des configurations réseau existantes. Par ailleurs, Kathará est un projet activement maintenu et déjà utilisé dans plusieurs universités pour l'enseignement des réseaux informatiques, y compris dans le cadre d'évaluations ou d'examens. Par exemple, des cours de réseaux à l'ETH Zurich, à l'University of Texas at Austin et à l'UCLouvain utilisent cet outil pour proposer des laboratoires réseau aux étudiants [3].

En combinant cet outil avec la plateforme INGInious, utilisée à l'UCLouvain pour la distribution et la correction automatique d'exercices, il devient possible de concevoir des activités pédagogiques interactives [4]. Cette approche s'inscrit dans la continuité de travaux récents sur les environnements d'apprentissage immersifs et les plateformes d'évaluation automatique destinées à l'enseignement de l'informatique [5]. L'objectif est de créer des exercices qui puissent être réutilisés sur le long terme et qui contribuent à enrichir l'écosystème pédagogique existant.

1.2 Objectifs

L'objectif principal de ce mémoire est de concevoir et de proposer de nouveaux exercices pratiques destinés aux étudiants en informatique, plus particulièrement dans les domaines des réseaux informatiques et de la cybersécurité. Ces exercices visent à appuyer les travaux pratiques existants en introduisant des activités centrées sur l'analyse, la compréhension et le débogage d'environnements réseau. Les thématiques abordées incluent notamment des concepts fondamentaux et plus avancés tels que le routage IP, le protocole BGP, OSPF et *Fast Reroute Using Segment Routing*.

Contrairement aux exercices classiques qui consistent souvent à implémenter une configuration complète à partir de zéro, les activités proposées dans ce travail

reposent principalement sur l'analyse de systèmes déjà existants. Les étudiants sont ainsi confrontés à des environnements réseau contenant des erreurs de configuration ou des comportements inattendus, qu'ils doivent identifier et corriger. Cette approche encourage le développement de compétences transversales telles que le diagnostic de problèmes, l'interprétation de configurations réseau et l'utilisation d'outils en ligne de commande pour analyser l'état du système.

Un autre objectif important de ce projet est de limiter les possibilités de triche ou d'utilisation abusive d'outils automatisés, notamment les assistants basés sur l'intelligence artificielle. En proposant des exercices nécessitant une interaction directe avec un environnement réseau et l'utilisation d'un terminal, il devient plus difficile de résoudre les problèmes par simple copie de réponses générées automatiquement. Cette approche favorise un apprentissage plus actif et encourage les étudiants à développer une compréhension réelle des mécanismes fondamentaux.

De plus, les exercices sont conçus de manière à permettre la génération de variantes différentes pour chaque étudiant. Cette personnalisation contribue à réduire les risques de partage de solutions entre étudiants tout en conservant des objectifs pédagogiques identiques.

Enfin, ce projet vise également à proposer un ensemble de ressources pédagogiques pérennes sous la forme d'un framework open source pour la conception d'exercices de réseaux. L'objectif est de faciliter la création, la modification et la réutilisation de laboratoires dans différents contextes pédagogiques, notamment dans les cours de réseaux informatiques et de cybersécurité. En s'appuyant sur des outils existants tels que Kathará pour l'émulation de topologies réseau et INGInious pour l'évaluation automatique, cette approche permet de construire une infrastructure d'exercices modulaire et extensible, pouvant être facilement intégrée dans l'infrastructure pédagogique existante. L'architecture générale de cette intégration est illustrée à la figure 3.3.

1.3 Structure du mémoire

Le reste de ce mémoire est organisé comme suit.

Le chapitre 2 présente l'état de l'art. Il introduit les concepts fondamentaux nécessaires à la compréhension du projet, passe en revue les outils existants de simulation et d'émulation réseau, et positionne la solution développée par rapport à ces alternatives.

Le chapitre 3 décrit la solution proposée. Il présente l'approche pédagogique retenue, les choix technologiques effectués, l'architecture du framework et le processus de génération des variants. Il détaille également les étapes de création d'un nouveau laboratoire.

Le chapitre 4 traite les aspects de sécurité et de scalabilité. Il analyse les vecteurs

de risque liés à l'exécution de code soumis par les étudiants, présente les mesures de protection mises en place, et évalue le comportement du système sous charge à travers des mesures de temps de correction et une analyse du mécanisme de scheduling.

Le chapitre 5 présente une analyse des retours recueillis concernant les exercices.

Le chapitre 6 identifie les axes d'amélioration et les perspectives d'évolution du framework, notamment en termes de mise à l'échelle et de support du mode examen.

Le chapitre 7 conclut ce mémoire en synthétisant les contributions réalisées, les limites identifiées ainsi que les perspectives d'évolution du framework proposé.

Chapitre 2

État de l'art

2.1 Concepts fondamentaux

Ce projet s'inscrit dans le domaine de l'enseignement des réseaux informatiques et de la cybersécurité. Afin de comprendre les mécanismes utilisés dans les exercices proposés, il est nécessaire de rappeler certains concepts fondamentaux liés aux réseaux informatiques. Cette section ne vise pas à fournir une description exhaustive de ces concepts, mais plutôt à présenter les éléments essentiels nécessaires à la compréhension du projet. Pour une présentation complète des principes fondamentaux des réseaux informatiques, le lecteur pourra se référer au syllabus accessible en ligne du cours de réseaux de bachelier [6].

2.1.1 Topologies réseau

Un réseau informatique peut être modélisé comme un ensemble de machines interconnectées échangeant des données à travers différents liens physiques. Dans ce contexte, on parle de *topologie réseau* pour décrire l'organisation structurelle des nœuds et des liens qui composent un réseau.

Les nœuds peuvent représenter différents types d'entités, tels que des routeurs, des switchs, des serveurs ou des machines hôtes. Les liens, quant à eux, représentent les connexions permettant l'échange de paquets entre ces nœuds. La topologie d'un réseau joue un rôle central dans son fonctionnement, notamment en ce qui concerne le routage du trafic et la résilience face aux pannes.

Dans le cadre de ce mémoire, les topologies réseau sont utilisées pour créer des environnements expérimentaux permettant aux étudiants d'analyser et de diagnostiquer différents scénarios de configuration.

2.1.2 Routage IP

Le routage constitue un mécanisme fondamental permettant aux paquets de circuler entre différentes machines à travers un réseau. Chaque routeur maintient une table de routage indiquant, pour chaque destination, le prochain saut (*next-hop*) vers lequel le paquet doit être transmis.

Deux grandes approches peuvent être distinguées : le routage statique et le routage dynamique. Le routage statique consiste à configurer manuellement les routes au sein des équipements réseau. Cette approche est simple à mettre en place mais ne passe pas à l'échelle dans des réseaux complexes ou en constante évolution, car toute modification de topologie nécessite une intervention manuelle sur chaque équipement [6].

À l'inverse, les protocoles de routage dynamique permettent aux routeurs d'échanger automatiquement des informations afin d'adapter leurs tables de routage. On distingue généralement deux catégories : les protocoles intra-domaine (IGP), utilisés à l'intérieur d'un système autonome, et les protocoles inter-domaine, utilisés pour l'interconnexion de systèmes autonomes distincts.

Parmi les protocoles intra-domaine, OSPF (*Open Shortest Path First*) et IS-IS (*Intermediate System to Intermediate System*) sont les plus répandus. Tous deux reposent sur un algorithme à état de liens (*link-state*) : chaque routeur diffuse des informations sur ses liens directement connectés, ce qui permet à l'ensemble des routeurs de construire une représentation commune de la topologie et de calculer les meilleurs chemins à l'aide de l'algorithme de Dijkstra [7], [8]. Ces deux protocoles sont exploités dans plusieurs laboratoires du framework développé dans ce mémoire.

Parmi les protocoles inter-domaine, BGP (*Border Gateway Protocol*) est le plus utilisé [9]. Contrairement aux protocoles intra-domaine, BGP n'échange pas d'informations sur la topologie du réseau mais des chemins vers des préfixes d'adresses. C'est sur ce protocole que repose l'interconnexion des réseaux constituant Internet.

2.1.3 Virtualisation et conteneurisation

Les technologies de virtualisation permettent d'exécuter plusieurs environnements isolés sur une même machine physique. Historiquement, cette isolation était principalement réalisée à l'aide de machines virtuelles complètes. Cependant, ces solutions peuvent être relativement lourdes en termes de consommation de ressources, car chaque machine virtuelle embarque son propre système d'exploitation complet.

La conteneurisation constitue une alternative plus légère reposant sur les mécanismes d'isolation du système d'exploitation. Les conteneurs permettent d'exécuter des applications dans des environnements isolés tout en partageant le noyau du système hôte. Cette approche offre plusieurs avantages, notamment une réduction

de l'utilisation des ressources, une meilleure reproductibilité des environnements et une rapidité de déploiement accrue [10]. En effet, une image Docker est définie par un fichier `Dockerfile` qui décrit l'ensemble des dépendances et configurations requises. Deux exécutions de la même image sur des machines différentes produisent un environnement identique, indépendamment du système hôte. Contrairement aux machines virtuelles, où la configuration d'un environnement peut nécessiter l'installation et la configuration manuelle de nombreux composants.

Ces images peuvent être distribuées et exécutées de manière identique sur différentes machines, garantissant ainsi que tous les utilisateurs disposent du même environnement d'exécution.

Cette reproductibilité est particulièrement importante dans un contexte pédagogique, où il est nécessaire de garantir que tous les étudiants disposent d'un environnement identique pour réaliser les exercices.

2.1.4 Docker et environnements isolés

Docker [10] est aujourd'hui l'une des plateformes de conteneurisation les plus largement utilisées. Elle repose sur le concept d'images, qui décrivent l'environnement logiciel nécessaire à l'exécution d'une application, et de conteneurs, qui constituent des instances exécutables de ces images.

Dans le contexte de ce mémoire, Docker permet de représenter chaque nœud d'une topologie réseau sous la forme d'un conteneur distinct. Chaque routeur ou machine hôte est ainsi exécuté dans un environnement isolé, ce qui permet de reproduire le comportement d'un réseau réel sur une seule machine.

Pour que des conteneurs puissent communiquer entre eux, Docker propose un mécanisme de réseaux virtuels (*Docker networks*). Chaque réseau virtuel constitue un segment réseau isolé auquel plusieurs conteneurs peuvent être connectés. Les conteneurs partageant un même réseau virtuel peuvent s'échanger des paquets comme s'ils étaient reliés par un lien physique, tandis que les conteneurs appartenant à des réseaux distincts sont isolés les uns des autres par défaut.

Ces conteneurs peuvent embarquer différents services réseau, tels que des implémentations de protocoles de routage ou des services comme des serveurs web ou DNS. Par exemple, l'image fournie par FRRouting [11] permet de déployer des routeurs capables d'exécuter des protocoles de routage avancés tels que BGP ou OSPF. De même, d'autres services peuvent être intégrés dans les conteneurs afin de reproduire différents scénarios réseau, comme un serveur DNS avec BIND9 [12] ou un serveur web léger avec Lighttpd [13].

2.1.5 Plateformes d'évaluation automatique

Les plateformes d'évaluation automatique jouent aujourd'hui un rôle central dans l'enseignement de l'informatique. Elles permettent de distribuer des exercices aux étudiants, de collecter leurs soumissions et d'exécuter automatiquement des tests afin de vérifier la validité des solutions proposées.

Dans ce contexte, la plateforme INGInious [4] constitue un outil particulièrement adapté pour l'enseignement. Elle permet de définir des exercices dans lesquels les étudiants soumettent du code ou des fichiers de configuration, qui sont ensuite évalués automatiquement à l'aide de scripts personnalisés. Les enseignants peuvent ainsi fournir un feedback immédiat aux étudiants tout en automatisant une grande partie du processus de correction.

2.2 Solutions existantes

La création de laboratoires réseau virtuels n'est pas un domaine nouveau. De nombreux outils ont été développés afin de simuler ou d'émuler des réseaux informatiques pour des usages pédagogiques, de recherche ou d'analyse de configurations.

On distingue généralement deux approches principales : la *simulation* réseau et l'*émulation* réseau. Les simulateurs modélisent le comportement d'un réseau à l'aide d'abstractions logicielles, tandis que les émulateurs exécutent de véritables logiciels réseau dans un environnement contrôlé reproduisant le fonctionnement d'un réseau réel.

Plusieurs outils populaires existent dans ce domaine, chacun présentant des avantages et des limitations.

2.2.1 Mininet

Mininet est un environnement d'émulation réseau historiquement très utilisé dans la recherche [14]. L'outil permet de créer rapidement des topologies réseau virtuelles sur une seule machine en s'appuyant sur les mécanismes d'isolation fournis par le noyau Linux.

Plus précisément, Mininet repose principalement sur l'utilisation de *Linux network namespaces*. Chaque nœud du réseau est représenté par un namespace isolé contenant sa propre pile réseau. Les liens entre les nœuds sont simulés à l'aide d'interfaces virtuelles (*veth pairs*) connectées entre namespaces. Cette approche permet d'exécuter de véritables applications réseau tout en simulant une topologie complexe. Bien que Mininet ait été largement adopté dans le monde académique, le développement du projet est aujourd'hui à l'arrêt depuis plusieurs années. En effet, la dernière release date de février 2021.

2.2.2 IPMininet

IPMininet est une extension de Mininet développée à l'UCLouvain, ajoutant le support natif des protocoles de routage réels via FRRouting, notamment OSPF, BGP et IS-IS [15], [16]. Elle expose une API Python permettant de décrire des topologies et leurs configurations protocolaires, ce qui facilite la création d'exercices.

Cependant, IPMininet hérite des limitations de Mininet : les nœuds partagent le système de fichiers hôte, ce qui offre une isolation moindre par rapport à une approche par conteneurs. Par ailleurs, le projet Mininet étant à l'arrêt depuis 2021, IPMininet suit une trajectoire incertaine sur le long terme. Enfin, comme Mininet, IPMininet ne propose pas de mécanisme de correction automatique ni de support natif pour la distribution d'exercices à des étudiants.

2.2.3 Klonet

Klonet est un système d'émulation réseau à grande échelle présenté lors de la conférence NSDI 2024, conçu pour déployer des topologies de plusieurs milliers de nœuds en distribuant des conteneurs sur un cluster de machines physiques [17]. Il vise à résoudre les problèmes de scalabilité des émulateurs existants tout en conservant l'exécution de véritables logiciels réseau.

Bien que Klonet démontre que l'émulation réseau par conteneurs reste un domaine de recherche actif, il est orienté exclusivement vers la recherche et l'expérimentation à grande échelle. Son infrastructure requiert un cluster de machines, ce qui le rend inadapté à un déploiement pédagogique sur les machines personnelles des étudiants. Il ne propose pas non plus de mécanisme de correction automatique ni de distribution d'exercices.

2.2.4 GNS3

GNS3 est une plateforme largement utilisée pour l'émulation de réseaux informatiques [18]. L'outil permet d'exécuter de véritables images de systèmes réseau (par exemple Cisco IOS ou des distributions Linux) à l'intérieur de machines virtuelles ou de conteneurs.

Grâce à son interface graphique, GNS3 facilite la construction de topologies réseau et l'interconnexion de différents équipements. Les équipements émulés s'exécutent à l'aide de différents moteurs tels que Dynamips, QEMU ou Docker, permettant ainsi de reproduire le comportement réel d'équipements réseau.

Cependant, cette approche repose souvent sur l'utilisation d'images système relativement lourdes et peut nécessiter des ressources matérielles importantes. Cela peut limiter son utilisation dans des environnements pédagogiques nécessitant un

déploiement rapide sur les machines des étudiants ou sur des infrastructures limitées en ressources.

2.2.5 NS-3

NS-3 est un simulateur de réseaux largement utilisé dans la recherche académique pour l'étude et l'évaluation de protocoles et d'architectures réseau [19]. Contrairement aux outils d'émulation, NS-3 repose sur une approche de simulation : le comportement du réseau est modélisé à l'aide de composants logiciels qui reproduisent les mécanismes des protocoles et des équipements réseau.

Dans NS-3, les topologies réseau sont décrites à l'aide de programmes écrits en C++ ou en Python. Les nœuds, les liens et les protocoles sont modélisés à travers des objets logiciels représentant les différentes couches du réseau. Cette approche permet de mener des expériences contrôlées et reproductibles, notamment pour évaluer les performances de nouveaux protocoles ou étudier le comportement de réseaux à grande échelle.

Cependant, NS-3 simule le fonctionnement du réseau plutôt que d'exécuter de véritables logiciels réseau. Les applications et protocoles utilisés sont des implémentations spécifiques au simulateur et ne correspondent pas toujours aux logiciels utilisés dans des environnements réels. Par conséquent, bien que NS-3 soit particulièrement adapté à l'analyse et à la recherche, il est moins approprié pour des laboratoires pédagogiques nécessitant l'utilisation d'outils et de configurations proches des environnements réseau réels.

2.2.6 Packet Tracer

Packet Tracer est un simulateur de réseau développé par Cisco et largement diffusé dans l'enseignement du domaine [20]. L'outil est intégré au programme Cisco Networking Academy et constitue une référence dans de nombreux cursus académiques pour l'apprentissage des concepts de commutation et de routage.

Packet Tracer repose sur une approche de simulation : il ne met pas en œuvre de véritables systèmes d'exploitation réseau, mais reproduit le comportement des équipements Cisco à travers une interface graphique. Cette approche le rend particulièrement accessible pour des exercices introductifs, mais limite son réalisme. Les commandes disponibles sont restreintes à un sous-ensemble de l'IOS Cisco, et les protocoles non supportés ou simplifiés ne peuvent pas être utilisés. De plus, l'outil est propriétaire et requiert un compte Cisco Networking Academy, ce qui restreint sa diffusion en dehors du cadre Cisco.

Dans un contexte d'enseignement avancé, notamment en master, ces limitations rendent Packet Tracer moins adapté que des solutions utilisant de véritables logiciels réseau. De plus, son caractère propriétaire et sa dépendance à l'écosystème Cisco

(*vendor lock-in*) peuvent rendre plus difficile l'adaptation des compétences acquises à d'autres environnements.

2.2.7 Containerlab

Containerlab est un outil relativement récent permettant de déployer des topologies réseau basées sur des conteneurs Docker [21]. Il est principalement utilisé dans des contextes d'ingénierie réseau, d'expérimentation et de validation d'infrastructures.

L'outil permet de décrire une topologie réseau à l'aide de fichiers de configuration, généralement au format YAML, puis de déployer automatiquement les différents nœuds sous forme de conteneurs interconnectés. Cette approche permet de reproduire rapidement des environnements réseau complexes et facilite l'intégration de ces laboratoires dans des workflows d'automatisation ou d'intégration continue.

Containerlab est ainsi particulièrement adapté aux environnements de test, d'expérimentation et de validation de configurations réseau dans des contextes professionnels.

À titre d'exemple, Containerlab est utilisé à l'UCLouvain dans le cadre du projet du cours LINFO2142 (Computer networks : configuration and management).

2.2.8 Kathará

Kathará est un framework open-source permettant de créer et d'exécuter des laboratoires réseau virtuels basés sur des conteneurs (Docker par défaut, Kubernetes via Megalos pour les grands scénarios) [2]. Chaque nœud du réseau est représenté par un conteneur, ce qui permet de reproduire des environnements réseau complexes sur une seule machine.

Cette approche présente plusieurs avantages dans un contexte pédagogique : les topologies peuvent être décrites de manière simple, déployées rapidement et exécutées sur des machines disposant de ressources limitées. De plus, les environnements créés sont facilement reproductibles.

Kathará propose également une API Python permettant de générer des topologies réseau. Cette fonctionnalité facilite la création d'exercices et permet d'automatiser la génération de différents scénarios.

L'un des composants particulièrement intéressants de cet outil est le *Kathara Lab Checker*, qui permet de définir des scénarios de tests automatisés pour vérifier le comportement d'une topologie réseau.

Ces tests peuvent être définis à l'aide de fichiers de configuration, par exemple en YAML.

Exemple de test automatisé

Le fichier `correction.yaml` définit les tests exécutés par le *Kathara Lab Checker* pour valider la configuration du laboratoire. Un extrait est présenté ci-dessous :

```yaml

test:
  requiring_startup:
    - h1
  ip_mapping:
    h1: { eth0: 10.0.1.2/24 }
    r1: { eth0: 10.0.12.1/30, eth1: 10.0.1.1/24 }
  reachability:
    h1:
      - 10.0.2.2
  custom_commands:
    r1:
      - command: "vtysh -c 'show ip bgp' | grep -E '^*'"
        output: "..."
```

Plusieurs catégories de tests sont disponibles : la vérification des adresses IP assignées aux interfaces (`ip_mapping`), la connectivité entre nœuds (`reachability`), ou encore l'état des démons réseau au démarrage (`requiring_startup`).

La catégorie `custom_commands` est particulièrement intéressante : elle permet d'exécuter une commande arbitraire sur un nœud et d'en vérifier la sortie par correspondance d'expression régulière (regex). Cette flexibilité est déterminante dans un contexte pédagogique, car elle permet de tester des propriétés spécifiques aux protocoles utilisés, telles que l'état des sessions BGP, le contenu des tables de routage, sans dépendre d'une mise à jour de l'outil. En pratique, cette catégorie est celle qui offre le plus de liberté pour concevoir des tests précis et adaptés à chaque laboratoire.

Ces mécanismes permettent d'automatiser la validation des exercices et constituent le socle de l'intégration avec INGInious.

2.2.9 Comparaison des outils

Le tableau 2.1 présente une comparaison simplifiée de plusieurs outils utilisés pour la simulation, l'émulation ou l'analyse de réseaux informatiques.

Outil	Type	Vérif. auto.[a]	Ressources[b]	Usage principal[c]
Mininet [14]	Émulation	Non	Faibles	Recherche et enseignement
IPMininet [16]	Émulation	Non	Faibles	Recherche, protocoles IP réels
Klonet [17]	Émulation	Non	Élevées	Recherche à grande échelle
GNS3 [18]	Émulation	Non	Élevées	Enseignement réseau proche du réel
NS-3 [19]	Simulation	Non	Moyennes	Recherche protocoles
Packet Tracer [20]	Simulation	Limitée	Faibles	Enseignement Cisco
Containerlab [21]	Émulation	Non	Faibles	Tests et expérimentation
Kathará [22], [23]	Émulation	Oui	Faibles	Enseignement réseau automatisable

[a] Présence d'un mécanisme intégré de validation automatique du comportement réseau.
[b] Besoins estimés en mémoire et CPU pour une topologie de 5 à 10 nœuds, d'après les documentations et publications officielles de chaque outil.
[c] Usage déclaré par les auteurs dans leurs publications ou documentations respectives.

TABLE 2.1 – Comparaison des outils de simulation et d'émulation réseau.

2.2.10 Limites des solutions existantes

L'analyse des outils présentés révèle plusieurs limitations communes, qui motivent l'approche retenue dans ce mémoire.

Absence de correction automatique intégrée. La majorité des outils d'émulation et de simulation ne proposent pas de mécanisme natif pour valider automatiquement le comportement d'un réseau configuré par un étudiant. Des outils comme Mininet, GNS3 ou Containerlab se concentrent sur le déploiement et l'exécution de la topologie, laissant à l'enseignant la charge de concevoir manuellement une procédure de vérification. Cette absence est particulièrement problématique dans un contexte pédagogique à grande échelle, où la correction manuelle de dizaines de configurations réseau n'est pas soutenable.

Kathará constitue une exception notable grâce au *Kathara Lab Checker*, qui fournit un mécanisme de vérification automatique du comportement réseau. C'est

cette propriété qui motive son choix comme environnement central du framework présenté dans ce mémoire.

Compromis entre réalisme et légèreté. Les outils existants se répartissent entre deux extrêmes. D'une part, les environnements très réalistes tels que GNS3 exécutent de véritables images système mais nécessitent des ressources matérielles importantes, ce qui complique leur déploiement sur les machines personnelles des étudiants. D'autre part, les simulateurs légers tels que NS-3 ou Packet Tracer offrent un déploiement aisé mais au prix d'une abstraction qui éloigne l'étudiant des outils et commandes réels utilisés en production.

Intégration limitée avec les plateformes d'évaluation. Les outils recensés ne proposent pas d'intégration native avec une plateforme d'évaluation automatique. L'articulation entre l'environnement d'émulation et la distribution, la collecte et la correction des exercices doit donc être entièrement construite par l'enseignant, ce qui représente un coût de mise en place significatif et freine l'adoption de ces outils dans un cadre universitaire.

Absence de mécanisme de différenciation par étudiant. Aucun des outils analysés ne propose nativement la génération automatique de variantes d'un même exercice, pourtant essentielle pour garantir l'intégrité académique lorsque plusieurs étudiants travaillent sur une même thématique. Cette fonctionnalité doit également être développée sur mesure.

Approche pédagogique par construction plutôt que par diagnostic. Enfin, les solutions existantes sont majoritairement pensées pour construire une topologie et en observer le comportement, plutôt que pour analyser et diagnostiquer une configuration préexistante. Or, l'apprentissage par le diagnostic (*learning by troubleshooting*) est reconnu comme une approche particulièrement efficace pour développer une compréhension profonde des mécanismes importants [24], d'autant plus lorsqu'elle s'appuie sur des simulations interactives combinées à un feedback et à un soutien pédagogique progressif (*scaffolding*) appropriés [25]. Ce positionnement pédagogique est au cœur du framework présenté dans ce mémoire.

2.3 Positionnement de notre solution

Dans le cadre de ce mémoire, le choix s'est porté sur l'utilisation de Kathará comme environnement principal pour la création de laboratoires réseau.

Plusieurs raisons motivent ce choix. Tout d'abord, Kathará permet de créer des topologies réseau complexes tout en restant relativement léger grâce à l'utilisation de conteneurs Docker. Cette approche facilite le déploiement des laboratoires sur les machines des étudiants.

Ensuite, la présence de l'outil *Kathara Lab Checker* constitue un avantage majeur. Celui-ci permet de définir des tests automatisés vérifiant la configuration et le comportement du réseau simulé. Ces tests peuvent être directement intégrés dans un processus d'évaluation automatique.

Par ailleurs, le projet Kathará est activement maintenu et son équipe de développement s'est montrée réactive tout au long de la réalisation de ce travail. Des échanges ont notamment permis de signaler des anomalies, de proposer des améliorations et de contribuer à l'ajout de fonctionnalités. [1].

Enfin, l'intégration avec la plateforme INGInious permet de proposer des exercices interactifs dans lesquels les étudiants déploient et analysent des topologies réseau tout en recevant un feedback automatique sur leurs réponses.

Cette combinaison d'outils offre ainsi un environnement pédagogique permettant aux étudiants de manipuler des réseaux réalistes tout en bénéficiant d'un système d'évaluation automatisé.

1. Allowing IPv6 on BGP neighbors checks : `https://github.com/KatharaFramework/kathara-lab-checker/pull/40`

Chapitre 3

Solution

3.1 Approche pédagogique

L'approche retenue place l'étudiant dans une démarche active d'analyse et de diagnostic. Plutôt que de construire une configuration réseau depuis zéro, l'étudiant reçoit un laboratoire préconfiguré contenant des erreurs intentionnelles qu'il doit identifier et corriger. Cette approche, centrée sur le *troubleshooting*, favorise une compréhension approfondie des mécanismes réseau en confrontant l'étudiant à des comportements inattendus dans un environnement réaliste.

Un *variant* désigne une archive ZIP constituant l'instance de laboratoire assignée à un étudiant. Elle contient l'ensemble des fichiers nécessaires à l'émulation locale de la topologie réseau via Kathara, ainsi qu'un fichier de tests permettant à l'étudiant de valider localement ses corrections. Ce fichier de tests est volontairement partiel : il garantit la cohérence minimale du réseau corrigé, tout en invitant l'étudiant à l'enrichir avec ses propres cas de test.

L'introduction de variants répond à un objectif d'intégrité académique : chaque étudiant se voit assigner un variant distinct, présentant une combinaison d'erreurs et un schéma d'adressage qui lui sont propres. En supposant que les erreurs injectables sont de difficulté comparable, tout variant contenant le même nombre d'erreurs présente un niveau de difficulté équivalent. Ainsi, ils portent sur les mêmes thématiques et les solutions ne sont pas directement transférables d'un étudiant à l'autre.

Le variant est mis à disposition via la plateforme INGInious et téléchargeable directement depuis l'énoncé de la tâche. La génération, le contenu et l'assignation des variants sont décrits en détail à la section 3.3.2.

3.2 Choix technologiques

L'architecture repose sur un ensemble d'outils complémentaires, choisis pour leur adéquation avec les contraintes pédagogiques et techniques du projet.

Kathará constitue l'environnement principal d'émulation réseau. Basé sur des conteneurs Docker, il permet de déployer des topologies réseau complexes sur une machine locale sans nécessiter de ressources matérielles importantes.

Python est utilisé pour l'ensemble des scripts d'automatisation : génération des variants, construction des archives ZIP distribuées aux étudiants, et extraction du feedback à partir des résultats de tests.

Jinja2 est utilisé comme moteur de *templating* afin de générer dynamiquement les fichiers de configuration des conteneurs à partir de modèles paramétrés. Les données variables (adresses IP, état des liens, présence d'erreurs) sont définies dans des fichiers **YAML**, qui servent de source de données pour le rendu des templates.

INGInious constitue la plateforme d'évaluation automatique. Elle assure la distribution des variants, la réception des soumissions et l'exécution des tests de correction, tout en fournissant un retour structuré à l'étudiant.

3.3 Architecture

Cette section présente les différents composants de l'architecture de la solution et décrit leurs interactions, tant du point de vue de l'étudiant que de celui du professeur.

3.3.1 Cours INGInious

La plateforme INGInious sert de point d'entrée pour les étudiants. Le cours est accessible à l'adresse suivante `https://inginious.info.ucl.ac.be/course/tfe-vanhorenbeke`, ou depuis la liste des cours disponibles. Il regroupe l'ensemble des exercices réalisés et fournit le contexte nécessaire à leur compréhension.

Tâches INGInious

Chaque tâche correspond à un exercice de laboratoire et se présente différemment selon le rôle de l'utilisateur.

Du point de vue de l'étudiant, une tâche expose les éléments suivants :

— les objectifs du laboratoire et les thématiques abordées ;
— un schéma de la topologie réseau émulée ;
— un lien de téléchargement vers le variant qui lui est assigné ;
— des indications méthodologiques pour orienter le diagnostic ;
— un champ de soumission permettant de déposer l'archive ZIP corrigée.

Du point de vue de l'enseignant, la configuration d'une tâche comprend plusieurs composants :

Paramètres de base Le titre et l'énoncé de l'exercice sont définis ici. L'énoncé intègre, via une balise HTML, la logique d'assignation et d'affichage du variant propre à chaque étudiant. Concrètement, un script JavaScript embarqué dans l'énoncé exploite le nombre aléatoire généré par INGInious pour chaque étudiant afin de calculer un indice de variant. Un lien de téléchargement pointant vers l'archive `variant_i.zip` correspondante est alors généré dynamiquement et affiché à l'étudiant. Ce mécanisme garantit que l'assignation est déterministe. En effet, un même étudiant obtiendra toujours le même variant, tout en étant uniforme sur l'ensemble des variants disponibles. Le script complet est disponible à l'annexe A.1.

Environnement d'exécution L'environnement de grading utilisé est une image Docker spécifique au projet, dont le code source est disponible sur le dépôt GitHub du projet [26].

Sous-problèmes Une unique question est définie par tâche : la soumission de l'archive ZIP contenant le laboratoire corrigé par l'étudiant.

Fichiers de la tâche La tâche embarque plusieurs fichiers et dossiers dont les rôles sont décrits ci-dessous.

- `run` : script principal orchestrant l'ensemble du pipeline de correction lors de chaque soumission. Son fonctionnement est détaillé dans le paragraphe suivant.
- `originalLab.conf` : copie de référence du fichier de topologie original. Avant l'exécution des tests, ce fichier est restauré dans le laboratoire soumis, neutralisant toute modification non autorisée de la structure du réseau. Ce mécanisme constitue la principale protection anti-triche : *Kathara Lab Checker* ne testant que les équipements déclarés dans ce fichier, sa suppression permettrait d'obtenir artificiellement un score de 100 %.
- Dossier `corrections/` : contient les fichiers de tests spécifiques à chaque variant. Le script de correction sélectionne automatiquement le fichier correspondant au variant assigné à l'étudiant.
- Dossier `public/` : contient les ressources statiques affichées dans l'énoncé (images, schémas de topologie) ainsi que le dossier `variations/` hébergeant les archives ZIP distribuées aux étudiants.
- Dossier `lang/` : voir section Multilingue 3.3.1.

Pipeline de correction Lors de chaque soumission, un pipeline de correction est automatiquement déclenché. Celui-ci se déroule en quatre phases principales, illustrées à la figure 3.1.

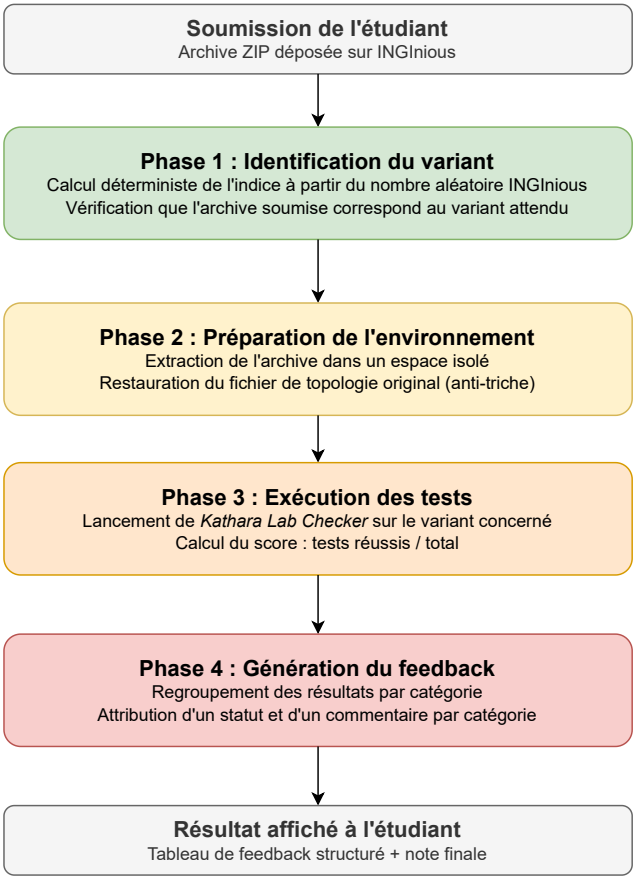


FIGURE 3.1 – Pipeline de correction par le script `run`

Phase 1 : identification du variant. L'indice du variant est calculé de manière déterministe à partir du nombre aléatoire associé au compte de l'étudiant sur la plateforme, garantissant qu'un même étudiant se voit toujours assigner le même variant. L'archive soumise est ensuite vérifiée : si elle ne correspond pas au variant attendu, la soumission est immédiatement rejetée et l'étudiant reçoit un lien de téléchargement direct vers l'archive correcte.

Phase 2 : préparation de l'environnement. L'archive soumise est extraite dans un espace isolé, puis le fichier de topologie original est systématiquement restauré, écrasant toute version soumise par l'étudiant. Ce processus constitue le mécanisme de prévention de la triche décrit précédemment.

Phase 3 : exécution des tests. Les tests sont sélectionnés automatiquement en fonction du variant concerné et couvrent plusieurs propriétés du réseau : cohérence de l'adressage IP, connectivité entre les nœuds, état des sessions des protocoles de routage et comportement des équipements vis-à-vis du trafic. L'outil produit en sortie plusieurs fichiers CSV récapitulant les tests réussis, échoués et l'ensemble des tests exécutés. Un score est ensuite calculé comme le ratio du nombre de tests réussis sur le nombre total de tests définis pour ce variant.

Phase 4 : génération du feedback. Le fichier CSV produit est analysé afin de regrouper les tests par catégorie thématique, telles que l'adressage, le routage ou la connectivité. Pour chaque catégorie, un statut est attribué selon le taux de réussite : *succès*, *échec* ou *partiel*. Un commentaire explicatif est associé aux catégories en échec afin d'orienter l'étudiant sans lui révéler directement la solution. L'ensemble est présenté sous forme d'un tableau structuré, accompagné de la note finale. Un exemple de tableau de feedback tel qu'affiché à l'étudiant est disponible à l'annexe B.1.

Support multilingue INGInious propose un mécanisme de traduction permettant de localiser le feedback affiché à l'étudiant [27]. Le principe repose sur quatre étapes :

- le marquage des chaînes de caractères à traduire dans les scripts Python et les templates de feedback
- l'extraction de ces chaînes via `pybabel` vers un fichier `.po`
- la traduction manuelle des entrées
- la compilation vers un fichier binaire `.mo`

Chaque langue requiert un dossier dédié dans l'arborescence de la tâche, sous `lang/<locale>/LC_MESSAGES/`.

Dans le cadre de ce projet, l'hypothèse retenue est que chaque exercice est d'abord rédigé en anglais, puis traduit en français. Ce choix s'explique par le fait que les exercices sont destinés à des étudiants de master, pour lesquels l'anglais constitue la langue utilisée en cours. La structure de dossiers nécessaires sont inclus dans l'architecture de chaque tâche, de sorte qu'ajouter une nouvelle traduction se résume à remplir le fichier `.po` correspondant et à relancer `pybabel compile`. Un exercice a été entièrement traduit à titre de démonstration ; les autres disposent d'une architecture prête à accueillir des traductions supplémentaires.

Machine virtuelle

L'infrastructure d'exécution a été mise en place par Anthony Gégo, mainteneur du système INGInious à l'UCLouvain. Elle repose sur une machine virtuelle hébergeant un démon Docker dédié aux simulations Kathara. L'image Docker de grading embarque les outils nécessaires à l'exécution des tests et est configurée pour accéder à ce démon via SSH. La connexion à la machine virtuelle est assurée par la plateforme StudSSH.

Ce déploiement garantit l'isolation des simulations réseau vis-à-vis des autres cours hébergés sur la plateforme, tout en conservant une transparence d'utilisation : les commandes Kathara exécutées dans le conteneur de grading sont identiques à celles disponibles sur la machine locale de l'étudiant.

Lorsqu'un étudiant soumet son archive, INGInious instancie un conteneur de grading qui exécute le script `run` de la tâche concernée, en déléguant les opérations d'émulation au démon Docker distant. Un schéma de l'architecture est repris en section 3.3.3 et une analyse sur la sécurité de ces machines est disponible à la section 4.2.

3.3.2 Génération des variants

La génération des variants repose sur une logique de *templating* permettant de produire automatiquement un ensemble de laboratoires distincts à partir d'une base commune paramétrable. Deux types de fichiers sont utilisés :

- des fichiers **YAML**, définissant les données d'entrée du laboratoire (topologie, adressage, protocoles, erreurs injectables) ;
- des fichiers **Jinja2** (`.j2`), constituant les templates des fichiers de configuration des conteneurs, avec des espaces réservés pour les valeurs variables.

Leur combinaison est orchestrée par le script `build_variants.py`, dont le pipeline est illustré à la figure 3.2.

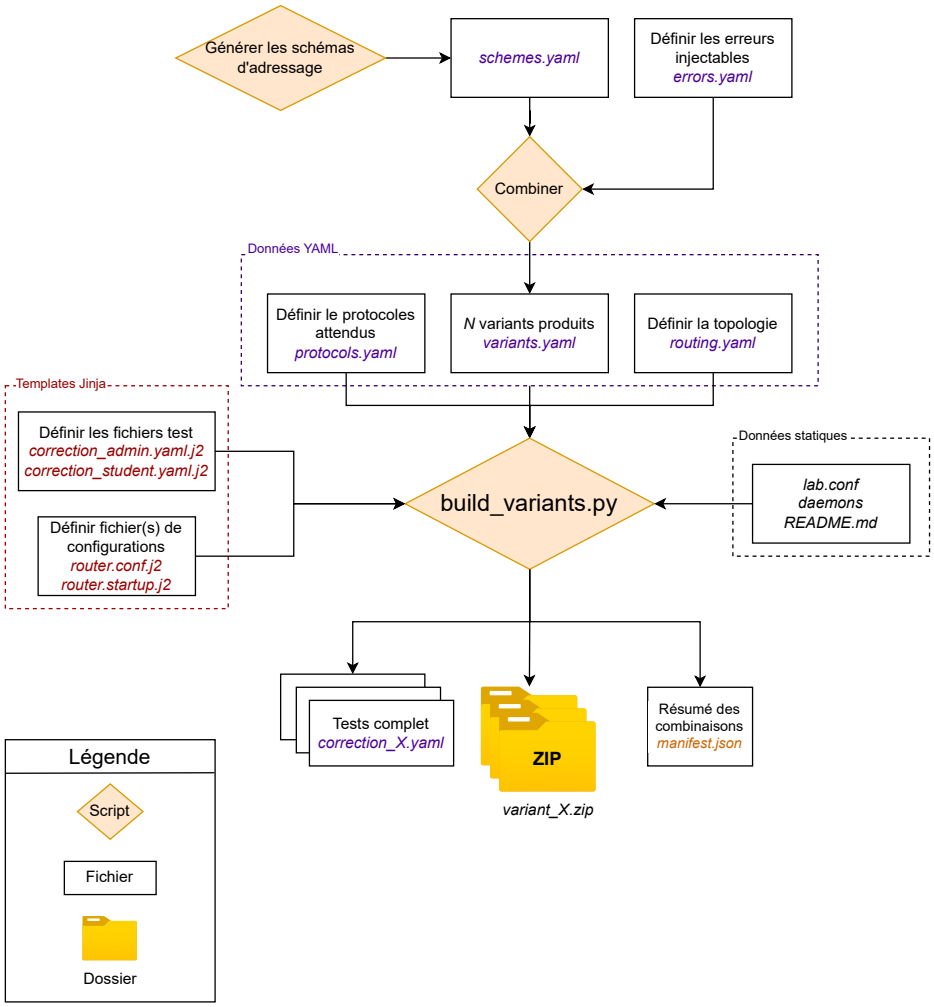


FIGURE 3.2 – Pipeline de génération des variants : les fichiers d'entrée YAML sont combinés avec des templates Jinja2 afin de produire automatiquement les configurations des laboratoires Kathara. Le script complète ces templates pour chaque variant, injecte les erreurs sélectionnées et génère à la fois les tests associés ainsi que l'état erroné (fourni à l'étudiant). L'ensemble est ensuite compressé sous forme d'archives ZIP distribuées via INGInious.

Fichiers d'entrée

Le modèle de laboratoire est défini par quatre fichiers YAML distincts.

— `routing.yaml` : décrit la topologie réseau, à savoir les nœuds, les liens, les interfaces et les services activés sur chaque équipement.

— `schemes.yaml` : spécifie les schémas d'adressage IP attribués à chaque variant. Ces schémas peuvent être définis manuellement ou générés automatiquement à l'aide d'un script dédié. Ce script analyse le fichier `lab.conf` afin d'en extraire la liste des nœuds et des domaines de collision, puis génère plusieurs plans d'adressage cohérents en IPv4 ou en IPv6 selon la configuration souhaitée. Les sous-réseaux sont attribués de manière aléatoire mais reproductible, garantissant que deux schémas distincts ne partagent pas le même espace d'adressage.
— `protocols.yaml` : contient les paramètres spécifiques aux protocoles de routage utilisés dans le laboratoire. Pour un laboratoire portant sur BGP, ce fichier précise par exemple le type de relation entre les systèmes autonomes (transit, peering), les voisins iBGP et eBGP, ou encore les interfaces participant à l'annonce de préfixes.
— `errors.yaml` : définit les erreurs injectables, c'est-à-dire les modifications volontairement introduites dans les configurations afin de constituer les exercices soumis aux étudiants.

Ces quatre fichiers constituent la base de données du laboratoire et délimitent l'espace des variants possibles.

Fichiers de templating Jinja2

Les fichiers Jinja2 constituent les modèles dans lesquels sont insérées les données issues des fichiers d'entrée. Trois templates sont définis.

— `router.conf.j2` : template du fichier de configuration FRRouting de chaque équipement réseau. Il produit un fichier de configuration standard dans lequel sont insérés dynamiquement les paramètres propres à chaque variant : identité du nœud, adresses des voisins BGP, politiques de routage, poids des liens, etc.
— `router.startup.j2` : template du fichier `.startup` utilisé par Kathara pour exécuter automatiquement des commandes au démarrage d'un conteneur. C'est dans ce fichier que sont générées les commandes d'assignation des adresses IP aux interfaces de chaque nœud.
— `correction.yaml.j2` : template du fichier de tests de correction. Deux versions sont produites à partir de ce template : une version *admin*, contenant l'ensemble des tests, utilisée lors de la correction automatique sur INGInious, et une version *student*, allégée, fournie à l'étudiant pour lui permettre de valider localement ses corrections.

La distinction entre ces deux versions répond à un objectif pédagogique précis. Si l'étudiant disposait de l'intégralité des tests, il pourrait identifier les erreurs à corriger par simple lecture du fichier, sans réelle analyse du comportement réseau. La version *student* est conçue de sorte à permettre à l'étudiant de détecter au

moins une erreur, tout en l'incitant activement à enrichir ce fichier avec ses propres cas de test. Cette démarche s'inscrit dans l'approche pédagogique par le diagnostic décrite à la section 3.1. Il est explicitement signalé à l'étudiant que le fichier local ne correspond pas à celui utilisé sur INGInious, afin d'éviter toute confusion si des tests supplémentaires échouent lors de la soumission.

Fichiers statiques

Certains fichiers nécessaires au laboratoire ne requièrent pas de templating et sont copiés tels quels dans chaque variant.

- `lab.conf` : fichier de topologie utilisé par Kathara pour instancier le laboratoire. Il décrit les nœuds du réseau, leurs interconnexions et les images Docker qui leur sont associées. Kathara s'appuie sur ce fichier, conjointement avec les fichiers `.startup`, pour déployer et configurer automatiquement les conteneurs au lancement du laboratoire.
- Fichiers de configuration des services : certains équipements requièrent des fichiers de configuration supplémentaires indépendants de l'adressage ou du protocole. Par exemple, le fichier `daemons` de FRRouting spécifie les démons réseau activés sur un routeur (BGP, OSPF, etc.).

Processus de génération

Pour chaque variant, le script résout d'abord l'état réseau attendu en combinant les données d'entrées. Une copie de cet état est ensuite modifiée par injection des erreurs définies pour ce variant, produisant ainsi l'état réseau abstrait que l'étudiant devra corriger. Les templates Jinja2 sont alors instanciés à partir de ces deux états : les fichiers de configuration (`.startup`, `frr.conf`) reflètent l'état erroné soumis à l'étudiant, tandis que le fichier `correction_X.yaml` est généré à partir de l'état attendu et servira de référence lors de la correction automatique. L'ensemble des fichiers produits est finalement assemblé en une archive `variant_X.zip`, prête à être distribuée via INGInious. Enfin, un dernier fichier, `manifest.json`, servant de récapitulatif des combinaisons est généré. Un exemple est donné dans l'annexe A.2.

Le nombre de variants N dépend de quatre paramètres : le nombre de schémas d'adressage distincts S, le nombre d'erreurs injectables E, et l'intervalle $[k_{\min},\ k_{\max}]$ définissant le nombre d'erreurs par variant. La formule générale est la suivante :

$$N = S \times \sum_{i=k_{\min}}^{k_{\max}} \binom{E}{i} \tag{3.1}$$

où $k_{\min}$ et $k_{\max}$ correspondent respectivement aux paramètres `--min-errors` et `--max-errors` du script de génération des combinaisons. Les variants sans erreur

sont explicitement exclus en imposant $k_{\min} \geq 1$: un laboratoire sans erreur n'aurait aucun intérêt pédagogique, l'étudiant n'ayant rien à corriger. Ces paramètres permettent de contrôler à la fois le nombre et la nature des variants générés. Toutefois, en pratique, un contrôle plus fin est souvent souhaitable, par exemple en imposant la présence d'un certain type d'erreur dans chaque variant afin d'en garantir la diversité. Cette contrainte s'est révélée utile pour homogénéiser les variants produits. Une discussion plus détaillée de ces aspects est proposée au chapitre consacré aux perspectives 6.

Le tableau 3.1 illustre l'évolution de N en fonction de $k_{\max}$, pour les paramètres du laboratoire BGP, soit $S = 4$ schémas d'adressage, $E = 8$ erreurs injectables et $k_{\min} = 1$:

$k_{\max}$	Combinaisons d'erreurs	N
1	$\binom{8}{1} = 8$	$4 \times 8 = 32$
2	$\binom{8}{1} + \binom{8}{2} = 36$	$4 \times 36 = 144$
3	$\binom{8}{1} + \binom{8}{2} + \binom{8}{3} = 92$	$4 \times 92 = 368$

TABLE 3.1 – Nombre de variants générés en fonction de $k_{\max}$, pour $S = 4$, $E = 8$ et $k_{\min} = 1$ (paramètres du laboratoire BGP)

Le paramètre $k_{\max}$ constitue un levier de dimensionnement important. En effet, augmenter $k_{\max}$ d'une unité revient à ajouter toutes les combinaisons de $k_{\max}$ erreurs parmi E, ce qui produit une croissance rapide du nombre de variants. À titre d'illustration, passer de $k_{\max} = 1$ à $k_{\max} = 2$ fait passer N de 32 à 144, soit un facteur 4,5 pour les paramètres du laboratoire BGP.

Ce framework permet ainsi de générer un grand nombre de variants à partir d'un ensemble de fichiers de templates, avec un effort de configuration réduit. En pratique, afin de garantir un niveau de difficulté homogène entre les étudiants, il est recommandé d'imposer $k_{\min} = k_{\max}$, de sorte que chaque variant contienne exactement le même nombre d'erreurs. Une disparité introduirait en effet une inégalité difficilement justifiable dans un contexte d'évaluation.

Extensibilité de l'architecture

L'architecture de templating présentée dans cette section s'est avérée maintenable et réutilisable au fil de la création des laboratoires. L'ajout d'un nouveau type d'erreur requiert trois modifications ciblées et bien délimitées : la définition de

l'erreur dans `errors.yaml`, son traitement dans le script de génération et l'association d'un feedback explicatif dans le pipeline de correction. De même, la création d'un nouveau laboratoire se résume à fournir les quatre fichiers d'entrée et les templates correspondants, sans modifier l'infrastructure existante. Cette séparation entre données, templates et logique de génération constitue un atout majeur pour la pérennité et l'extension du framework.

3.3.3 Architecture d'exécution

Lorsqu'un étudiant soumet une archive, le traitement traverse plusieurs niveaux successifs, illustrés à la figure 3.3.

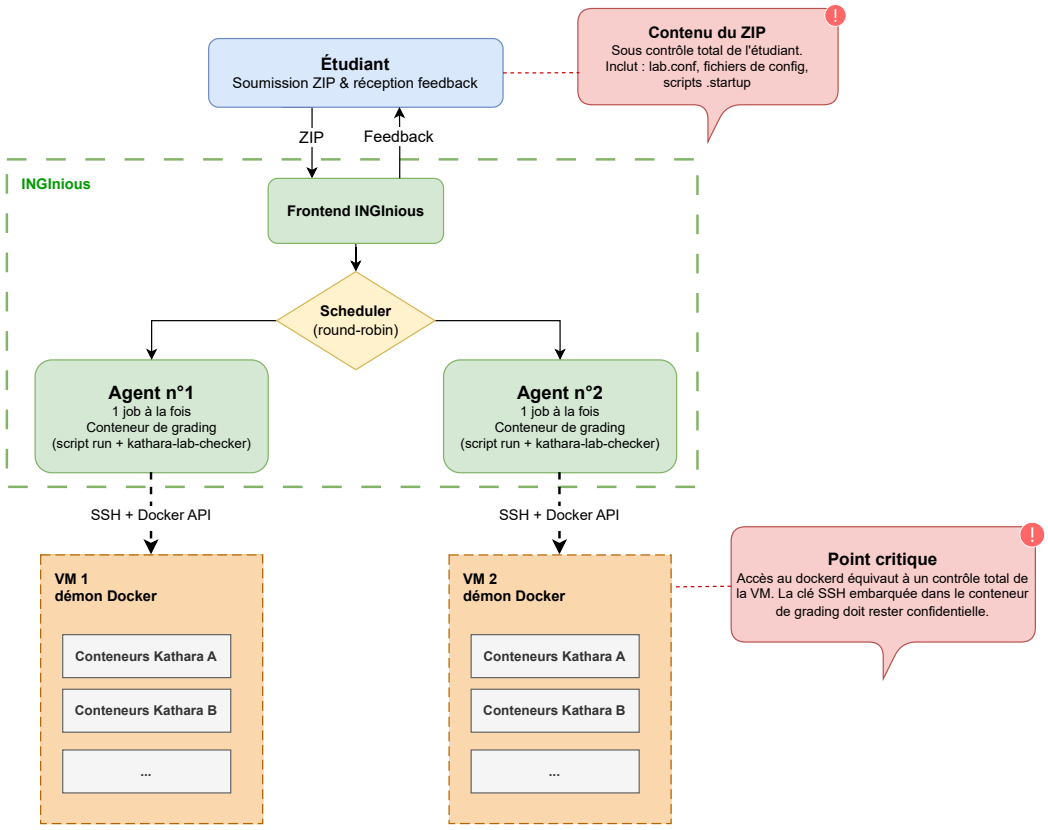


FIGURE 3.3 – Architecture d'exécution d'une soumission, de l'étudiant aux conteneurs Kathara sur les VM distantes

L'archive ZIP est d'abord reçue par la plateforme INGInious, qui distribue le job à l'un des agents disponibles disposant de l'environnement Kathara. Dans le

déploiement actuel, deux agents dédiés sont configurés pour cet environnement. Chacun n'accepte qu'un seul job à la fois et est associé à sa propre VM [1] utilisant Docker. La répartition des jobs entre agents s'effectue selon un mécanisme de round-robin.

L'agent sélectionné instancie un conteneur de grading dédié à la correction. Ce conteneur exécute le script `run` décrit à la figure 3.1. L'émulation Kathara elle-même ne se déroule pas à l'intérieur de ce conteneur : Kathara requiert des privilèges d'exécution élevés qui ne sont pas disponibles dans l'environnement INGInious et qui, s'ils l'étaient, compromettraient la sécurité du conteneur de grading en donnant à l'étudiant un levier sur l'environnement de correction. L'émulation est donc déléguée à un démon Docker hébergé sur la VM distante associée à l'agent [28], accessible via SSH à l'aide d'une clé privée embarquée dans l'image de grading. Les conteneurs Kathara sont ainsi créés sur la VM distante, et non dans l'environnement INGInious.

3.4 Implémentation

3.4.1 Créer un nouveau laboratoire

Cette section décrit les étapes nécessaires à la création d'un nouveau laboratoire dans le framework. L'ordre proposé vise à limiter les allers-retours entre les différentes étapes : chaque étape s'appuie sur les sorties de la précédente.

Étape 0 — Réflexion préliminaire. Avant toute implémentation, il est recommandé de définir clairement le sujet du laboratoire, les concepts réseau ciblés et les erreurs que l'étudiant devra diagnostiquer. Cette réflexion en amont facilite la conception de la topologie et du jeu d'erreurs, et évite de retravailler les fichiers de templates en cours de développement.

Étape 1 — Définir la topologie. Écrire le fichier `lab.conf` décrivant les nœuds, les liens et les images Docker associées. Ce fichier constitue la base statique du laboratoire et est distribué tel quel dans chaque variant.

Étape 2 — Compléter les fichiers YAML d'entrée. Renseigner `routing.yaml` avec les interfaces, les daemons et les paramètres de chaque nœud, puis `errors.yaml` avec les erreurs injectables. Si de nouveaux types d'erreurs sont définis, le script `build_variants.py` doit être étendu en conséquence pour gérer

1. Machine virtuelle KVM/QEMU : Ubuntu, 4 vCPU Intel Broadwell et 16 Go de RAM.

leur injection, et le script d'extraction du feedback doit être mis à jour pour les catégoriser et leur associer un commentaire.

Étape 3 — Générer les fichiers dérivés. Les fichiers `schemes.yaml` et `variants.yaml` peuvent être générés automatiquement via les scripts auxiliaires fournis. La génération des schémas d'adressage se fait à partir du `lab.conf`, tandis que les combinaisons de variants sont produites en fonction des paramètres `-min-errors` et `-max-errors`.

Étape 4 — Écrire les templates Jinja2. Compléter les fichiers `router.startup.j2`, `router.conf.j2` et `correction.yaml.j2` afin de refléter la topologie et les protocoles utilisés. Le fichier `correction.yaml.j2` doit inclure les tests permettant de détecter chacune des erreurs définies à l'étape 2.

Étape 5 — Générer les variants. Lancer `build_variants.py` pour produire les archives ZIP. Il est conseillé de générer en premier lieu un variant sans erreur, afin de valider que la topologie de base fonctionne correctement et que les tests passent tous. Une fois cette vérification effectuée, les variants avec erreurs peuvent être générés.

Le processus complet est illustré à la figure 3.4.

Intégration avec INGInious

Une fois les variants générés, la tâche INGInious peut être créée. Le script `run` fourni est conçu pour être réutilisé sans modification majeure : seules les variables de configuration en tête de fichier (chemins, URL de téléchargement, identifiant de la tâche) sont à adapter.

Configuration de la tâche. Créer une nouvelle tâche sur la plateforme INGInious et configurer l'environnement de grading comme indiqué à la section 3.3.1. Définir ensuite un sous-problème de type *téléchargement de fichier*, qui constituera le champ de soumission de l'archive ZIP par l'étudiant.

Copier le script de correction. Copier le script `run` dans les fichiers de la tâche et adapter les variables de configuration en tête de fichier : chemin vers les fichiers de correction, URL de base pour le téléchargement des variants, et identifiant du sous-problème de soumission.

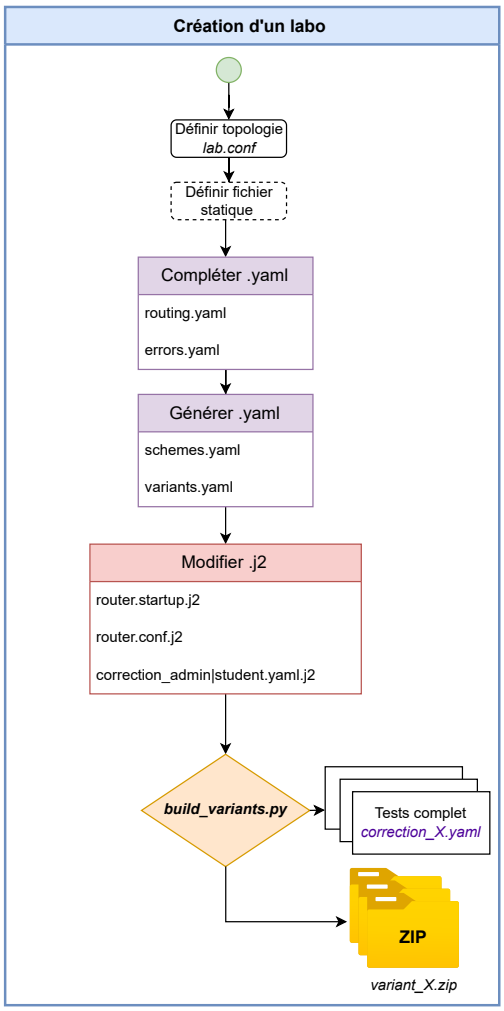


FIGURE 3.4 – Processus de création d'un nouveau laboratoire

Construire l'arborescence de fichiers. L'arborescence attendue par le script est la suivante :

- `corrections/` : contient les fichiers de tests complets générés par `build_variants.py`, un fichier par variant (`correction_admin_X.yaml`).
- `public/` : contient les ressources affichées dans l'énoncé (schémas de topologie, images) ainsi qu'un dossier `public/variations/` dans lequel sont déposées toutes les archives ZIP des variants.
- `originalLab.conf` : copie de référence du fichier de topologie, utilisée par le mécanisme anti-triche décrit au paragraphe 3.3.1.

Le workflow complet d'intégration est illustré à la figure 3.5.

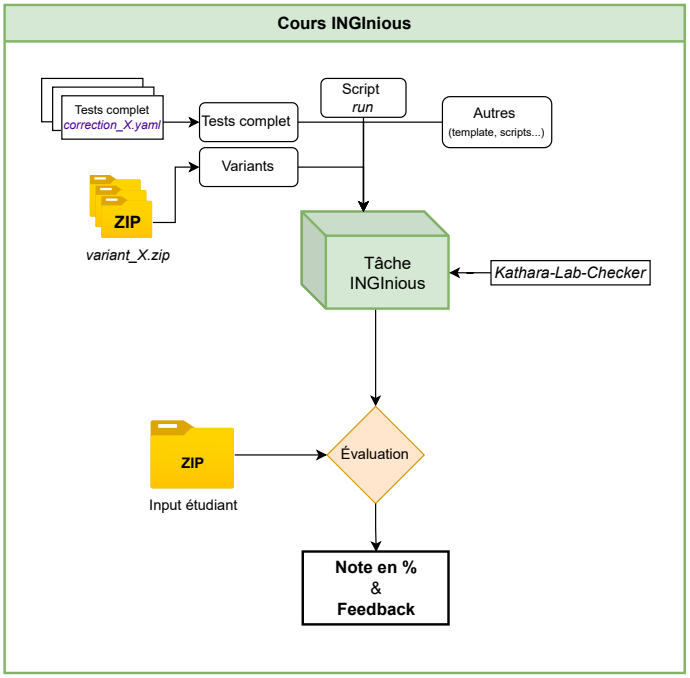


FIGURE 3.5 – Workflow d'intégration d'un laboratoire dans INGInious

3.4.2 Exemple de laboratoire

Laboratoire de cybersécurité : attaques réseau

Ce laboratoire illustre l'extensibilité du framework à des thématiques autres que le routage. Il porte sur trois attaques réseau classiques : le SYN Flood, le TCP Port Scan et la réflexion DNS. Contrairement aux laboratoires de routage, il repose sur

une configuration Kathará statique : la topologie est fixe et préconfigurée, l'objectif étant de se concentrer sur l'implémentation et la compréhension des attaques.

Ce laboratoire remplace directement une version précédente dans laquelle les étudiants devaient configurer manuellement leurs interfaces réseau, ajuster les règles de pare-feu et installer Scapy sur leur propre machine. Cette procédure était source d'erreurs fréquentes sans rapport avec les objectifs du cours. Désormais, l'étudiant télécharge l'archive, lance `kathara lstart` et se retrouve immédiatement dans un environnement opérationnel. Scapy, `tcpdump` et Wireshark sont préinstallés sur les nœuds concernés. La capture réseau s'effectue dans un environnement isolé, éliminant le bruit lié au trafic extérieur d'une machine personnelle.

Le laboratoire est décomposé en deux tâches INGInious : une tâche théorique avec des QCM sur les mécanismes des attaques et la topologie, et une tâche de correction automatique dans laquelle l'étudiant soumet un fichier `capture.pcap`. Le script de correction vérifie automatiquement que les trois attaques sont présentes dans la capture et génère un tableau de feedback structuré (annexe B.3).

Chapitre 4

Sécurité et scalabilité

L'architecture mise en place fait interagir plusieurs composants distincts, chacun introduisant ses propres considérations en terme de sécurité. Ce chapitre analyse l'architecture d'exécution présenté à la figure 3.3 afin de clarifier les zones de confiance, puis analyse les risques identifiés et leur traitement. Enfin, la scalabilité du système est discutée sur la base des observations effectuées lors des tests dans la section suivante 4.3.

4.1 Modèle de menace

L'analyse repose sur les **hypothèses** suivantes, qui délimitent le périmètre considéré :

- l'étudiant est considéré comme un attaquant potentiellement motivé, cherchant à tricher, à exfiltrer des informations ou à compromettre l'infrastructure ;
- l'étudiant contrôle entièrement le contenu de l'archive soumise, y compris les fichiers `.startup` qui sont exécutés au démarrage des conteneurs Kathara ;
- l'étudiant peut soumettre un fichier par tâche, en parallèle de soumissions d'autres étudiants.

L'**objectif** est qu'un étudiant ne doit pouvoir ni accéder aux soumissions d'autres étudiants, ni modifier le système de notation, ni exfiltrer des secrets de l'infrastructure, ni dégrader le service pour les autres utilisateurs.

4.2 Analyse de sécurité

4.2.1 Exécution de code malveillant fourni par l'étudiant

L'exécution des commandes se passent dans le container. Les fichiers `.startup` soumis par l'étudiant sont exécutés automatiquement par Kathara au démarrage des conteneurs. Il serait donc possible pour un étudiant d'y insérer des commandes arbitraires. Cette exécution reste cependant cantonnée à l'intérieur des conteneurs Kathara, eux-mêmes isolés par Docker sur la VM distante. Le script de correction, de son côté, se limite à invoquer *Kathara Lab Checker* et à traiter son résultat structuré : aucune commande fournie par l'étudiant n'est utilisée dans le conteneur de grading. De plus, le *parsing* réalisé qui attribue les points est basé sur le résultat de *Kathara Lab Checker*, donc un étudiant ne peut pas provoquer des erreurs via des caractères spéciaux. Cette séparation nette entre l'environnement qui exécute le code non fiable et celui qui orchestre la correction constitue la principale mitigation contre ce vecteur d'attaque.

4.2.2 Fuite d'information via le feedback

Le feedback affiché à l'étudiant ne doit jamais contenir la sortie brute des commandes exécutées, des logs du système, ni le contenu de `stderr` ou `stdout`. En effet, un étudiant malveillant pourrait alors orchestrer l'affichage de fichiers sensibles présents dans le conteneur de grading, notamment la clé SSH donnant accès au démon Docker. Dans l'implémentation actuelle, seuls les résultats structurés produits par *Kathara Lab Checker* sont remontés, puis agrégés par catégorie avant affichage. Cette propriété est simple à maintenir mais critique, et doit être explicitement vérifiée à chaque évolution du script de correction.

4.2.3 Intégrité de la topologie soumise

Comme détaillé à la section 3.1, le fichier `lab.conf` soumis par l'étudiant est systématiquement écrasé par une version de référence avant l'exécution des tests. Ce mécanisme empêche un étudiant d'obtenir artificiellement un score maximal en supprimant les équipements sur lesquels portent les tests.

4.2.4 Archives malveillantes

Les archives ZIP soumises sont extraites dans un espace isolé propre à chaque soumission. Concernant les archives de type *zip bomb*, leur extraction provoquerait le crash du conteneur de grading, aboutissant à une soumission considérée comme nulle sans impact sur le reste de l'infrastructure. Concernant les liens symboliques

ou chemins relatifs malveillants, ceux-ci ne peuvent pas sortir du système de fichiers du conteneur dans lequel l'extraction est effectuée, les vérifications des droits UNIX étant pleinement applicables dans cet environnement. Dans la mesure où seule la soumission de l'étudiant est injectée dans cet espace, les accès effectués restent a priori légitimes. Le mécanisme `run` fourni par INGInious peut par ailleurs être utilisé pour protéger des fichiers sensibles du conteneur de grading contre tout accès non autorisé depuis le code de correction via les vérifications des droits UNIX.

4.3 Analyse de scalabilité

4.3.1 Méthodologie de mesure

Le temps de correction de chaque soumission est mesuré directement dans le script `run` à l'aide d'un timestamp en millisecondes capturé au début et à la fin de l'exécution.

Afin d'évaluer le comportement du système sous charge, six comptes INGInious temporaires ont été utilisés pour simuler des soumissions simultanées. Cette approche a révélé que le temps de correction effectif ne varie pas avec le nombre de soumissions parallèles : c'est le temps d'attente en file qui augmente, chaque soumission étant mise en attente jusqu'à ce qu'un agent soit disponible. Les mesures du temps de correction se limitent donc à un étudiant soumettant plusieurs tâches différentes en parallèle. Ce point est approfondi dans les perspectives, section 6.3

4.3.2 Problème de concurrence identifié

Avant la mise en place de l'architecture de scheduling décrite à la figure 3.3, un problème de concurrence a été identifié lors de tests sans mécanisme de sérialisation. Lorsque deux étudiants soumettent la même tâche avec un intervalle inférieur à cinq secondes sur un même démon Docker, l'une des deux corrections échoue sans produire de résultat.

Cette erreur provient d'une limitation de Kathara dans un contexte de parallélisme sur un même démon Docker. Kathara déploie un laboratoire en plusieurs étapes non atomiques : création des réseaux Docker, création des conteneurs, attachement des conteneurs aux réseaux, puis démarrage. Lorsque deux corrections s'exécutent en parallèle sur le même démon, leurs opérations s'entrelacent durant cette phase critique. Un réseau créé par la première correction peut devenir inaccessible avant que ses conteneurs ne s'y attachent, en raison des interférences avec la seconde correction. Ce comportement constitue la motivation principale pour l'architecture de scheduling retenue.

En revanche, si l'intervalle entre deux soumissions est de l'ordre de cinq à dix

secondes, la phase critique de déploiement de la première correction est entièrement terminée avant que la seconde ne commence, et les deux corrections aboutissent correctement.

4.3.3 Solutions envisagées

Trois approches ont été explorées pour traiter ce problème de concurrence.

Verrou global applicatif. La première approche consistait à sérialiser complètement les corrections via un verrou applicatif posé dans le script de correction. Une seule correction s'exécute alors à la fois, les autres attendant leur tour. Cette approche est fiable mais introduit une file d'attente strictement séquentielle. Pour N étudiants soumettant simultanément une tâche dont la correction dure T secondes, le dernier étudiant attend $N \times T$ secondes au maximum. Pour $N = 20$ et $T = 45\,\text{s}$, cela représente une attente de quinze minutes, ce qui est pédagogiquement inacceptable.

Verrou limité à la phase critique. La seconde approche consistait à ne poser le verrou que pendant la phase de déploiement de Kathara, dont la durée effective est de cinq à sept secondes, en appliquant une marge de sécurité portant ce délai à $T_{\text{lock}} = 15\,\text{s}$. Pour $N = 20$ étudiants soumettant simultanément, l'attente maximale est :

$$T_{\text{attente}} = N \times T_{\text{lock}} + T_{\text{tests}} = 20 \times 15 + T_{\text{tests}} \tag{4.1}$$

soit environ cinq minutes au total, contre quinze minutes avec un verrou global.

Scheduling au niveau des agents INGInious. La troisième approche, finalement retenue, exploite le mécanisme de scheduling natif d'INGInious. L'environnement de grading Kathara est déployé sur deux agents distincts (`hyperion` et `theia`), chacun connecté à sa propre machine virtuelle Docker. Chaque agent n'accepte qu'une seule correction à la fois, et INGInious distribue les soumissions entre les agents disponibles selon un mécanisme de round-robin, comme illustré à la figure 3.3.

Cette architecture présente deux propriétés importantes. La sérialisation par agent garantit qu'aucune collision de concurrence ne peut se produire au sein d'un même démon Docker, sans nécessiter de verrou applicatif. La présence de deux agents permet par ailleurs à deux corrections de s'exécuter réellement en parallèle, chacune sur sa propre VM, doublant ainsi le débit de correction.

4.3.4 Analyse du temps de correction

Le temps de correction effectif a été mesuré pour chaque exercice en conditions normales (N=1 tâche en parallèle), avec trois répétitions par tâche. Les résultats sont présentés à la figure 4.1.

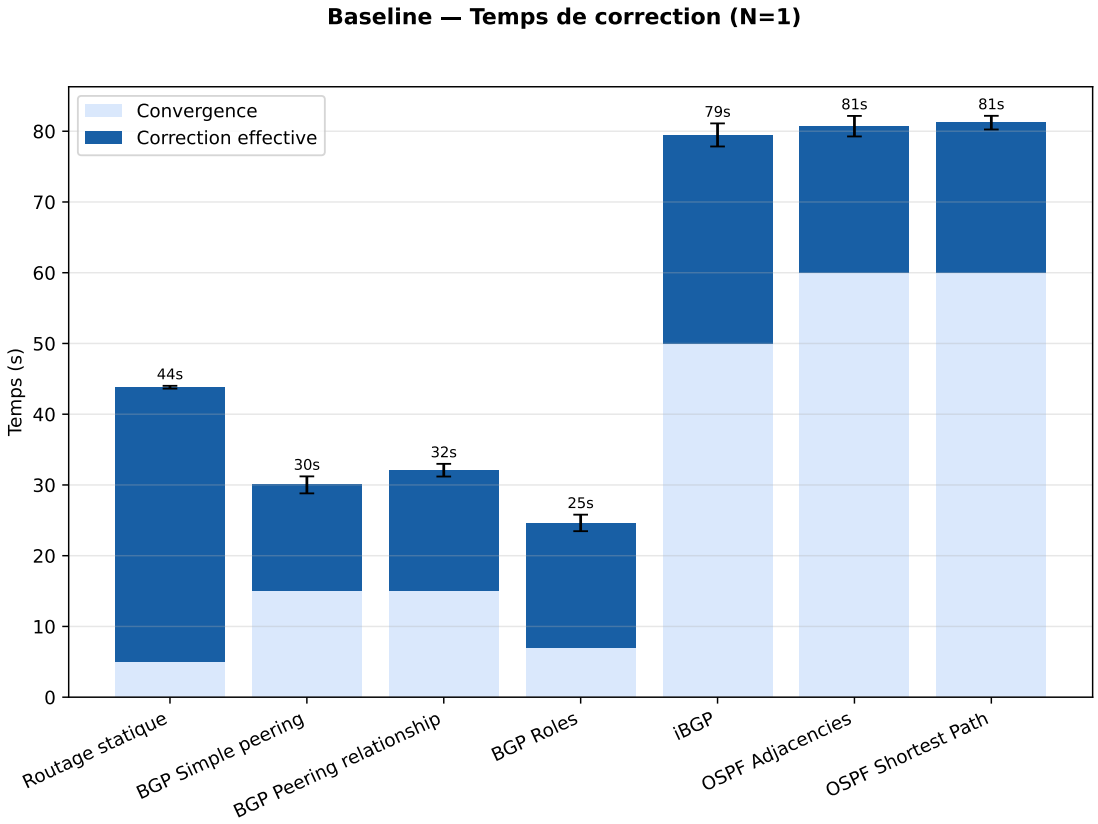


FIGURE 4.1 – Temps de correction par exercice (N=1, 3 répétitions). Les barres distinguent le temps de convergence réseau (fixe) du temps de correction effectif.

La variance entre répétitions est inférieure à 5 % pour toutes les tâches, ce qui confirme la reproductibilité du pipeline de correction. Les temps de correction mesurés sur INGInious sont identiques à ceux obtenus en exécution locale, ce qui indique l'absence d'opérations parasites introduites par l'infrastructure de grading.

Les temps de convergence varient selon les exercices pour deux raisons distinctes. D'une part, certains protocoles comme BGP et OSPF nécessitent un délai d'établissement des sessions avant que les tests puissent être exécutés : ce délai est intentionnel et fait partie de la configuration du laboratoire. D'autre part, le mécanisme de test de connectivité (`ping`) introduit un délai supplémentaire lorsque des destinations sont inaccessibles : le timeout par défaut est de 10 secondes

par destination non joignable, et les tests s'exécutant séquentiellement, plusieurs destinations inaccessibles peuvent cumuler ces délais [1].

C'est ce second mécanisme qui explique le temps total d'environ 44 secondes observé pour le premier exercice (routage statique), malgré sa simplicité relative. Le laboratoire iBGP présente quant à lui un temps de correction élevé pour les deux raisons combinées : le protocole iBGP nécessite un délai de convergence avant que les sessions soient établies, et des tests de connectivité sont également présents, dont les timeouts s'accumulent en cas d'erreurs dans la configuration soumise.

Il est à noter que ces délais ne se manifestent que lorsque la configuration soumise contient des erreurs. Pour une archive correctement corrigée, les destinations sont toutes joignables et aucun timeout n'est déclenché, ce qui réduit significativement le temps de correction effectif.

4.3.5 Analyse du scheduling

Afin de visualiser concrètement le comportement du scheduling, l'activité des deux VM a été enregistrée à intervalles réguliers (une seconde) pendant l'exécution simultanée de six soumissions. Le diagramme de Gantt résultant est présenté à la figure 4.2.

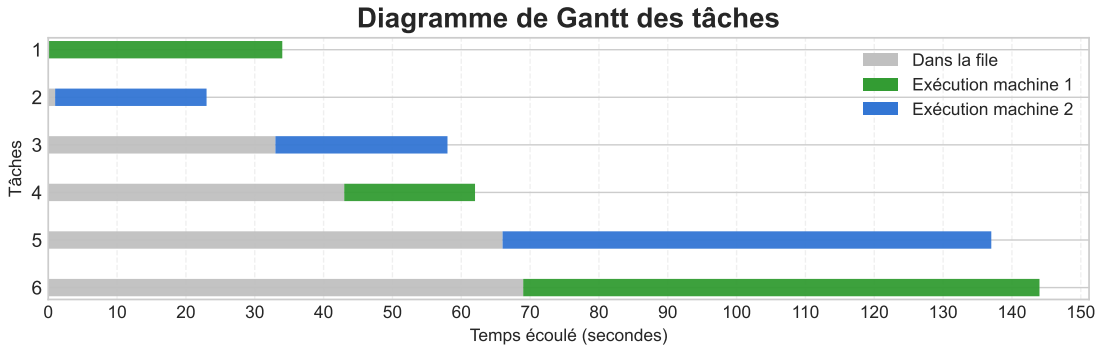


FIGURE 4.2 – Diagramme de Gantt de l'exécution de six soumissions simultanées.

Le diagramme confirme le bon fonctionnement du mécanisme de round-robin : les six tâches sont réparties équitablement entre les deux VM, avec exactement trois tâches par agent et au maximum une tâche active simultanément par VM. On observe également un délai d'environ dix secondes entre la fin d'une tâche et le début de la suivante sur la même VM. Ce délai s'explique principalement par la phase de nettoyage, qui inclut l'arrêt des conteneurs, la suppression des

1. Implémentation du test disponible à l'adresse : `https://github.com/KatharaFramework/kathara-lab-checker/blob/main/src/kathara_lab_checker/checks/ReachabilityCheck.py`

réseaux Docker et la libération des ressources, gérés par Kathara et le démon Docker distant. S'y ajoutent la coordination INGInious, avec le polling des agents et la synchronisation des statuts de soumission, ainsi que les coûts de communication via SSH entre le conteneur de grading et la VM distante. L'ensemble de ces étapes produit ainsi un intervalle quasi constant de dix secondes entre deux exécutions successives sur une même machine virtuelle.

La répartition de charge observée valide l'architecture retenue : en supprimant le partage d'un même démon Docker entre corrections concurrentes, le scheduling par agent élimine le risque de collision identifié à la section 4.3.3, sans nécessiter de mécanisme de synchronisation applicatif.

4.3.6 État du démon Docker entre exécutions

Une situation a été identifiée pouvant laisser des résidus persistants sur la VM distante. Lorsqu'un étudiant soumet une solution puis interrompt immédiatement le processus de correction, des conteneurs et des réseaux virtuels créés par Kathara peuvent persister sur la machine distante après l'arrêt de l'exécution. En l'absence d'un mécanisme de nettoyage explicitement déclenché lors de l'annulation, ces ressources deviennent orphelines.

Ces résidus n'ont pas d'impact sur les corrections en cours ou futures : chaque laboratoire Kathara opère avec des identifiants uniques, de sorte que les objets orphelins n'interfèrent pas avec le déroulement des soumissions suivantes. Ils constituent néanmoins une accumulation progressive de ressources inutilisées sur la VM distante.

Un mécanisme de nettoyage automatique a été mis en place pour traiter ce problème. Un service `systemd`, activé au démarrage de la VM, vérifie toutes les minutes l'état des conteneurs Docker actifs et supprime, ainsi que leurs réseaux associés, tout conteneur en cours d'exécution depuis plus de dix minutes. Cette durée est supérieure au temps de correction maximal observé, garantissant qu'aucune correction en cours n'est interrompue. Ce mécanisme assure ainsi un nettoyage régulier de la VM sans intervention manuelle.

Chapitre 5

Analyse des retours

5.1 Méthodologie de collecte

Les exercices ont été diffusés à un groupe restreint d'étudiants via une annonce sur Moodle dans le cadre du cours de réseaux de deuxième quadrimestre, ainsi que sur des canaux de communication étudiants informels. Cette phase constitue une évaluation préliminaire du framework, avec pour objectifs de valider la clarté des consignes, la faisabilité des exercices et la portabilité des laboratoires sur les machines personnelles des étudiants. Les retours ont été collectés via un formulaire Google Form. En parallèle, plusieurs assistants ont pu tester les exercices tout au long du développement et fournir un feedback technique précieux, notamment Anthony Doeraene pour les parties réseaux et Rémi Van Boxem pour la partie cybersécurité. Ce dernier s'est montré particulièrement intéressé par l'utilisation de Kathará pour simplifier les configurations réseau dans le cadre du cours de LINFO2347 *Computer System Security*.

Compte tenu du nombre limité de participants (trois étudiants), les conclusions de cette section sont à interpréter comme des observations préliminaires plutôt que comme des résultats statistiquement représentatifs.

5.2 Retours des étudiants

Portabilité et installation. Les premiers tests ont mis en évidence des difficultés lors de l'installation de *Kathara Lab Checker* sur les machines des étudiants, liées à des incompatibilités dans certaines versions de dépendances. Ces anomalies ont été documentées dans l'énoncé des exercices afin d'orienter les étudiants, et ont également été remontées à l'équipe de développement de l'outil via des rapports

sur le dépôt GitHub du projet [1]. Ces retours ont conduit à des corrections dans les versions suivantes de l'outil.

Clarté des consignes et faisabilité. Les retours indiquent que les consignes sont globalement comprises et que les exercices sont jugés accessibles. La topologie fournie et les indications méthodologiques ont été perçues comme suffisantes pour orienter le diagnostic sans révéler directement les erreurs à corriger.

Utilisation d'un outil d'IA générative. Une observation notable a été faite lors des tests : une étudiante a tenté de résoudre le premier exercice en transmettant directement les fichiers de configuration et l'énoncé à un assistant d'IA générative, sans chercher à analyser le problème par elle-même. Cette approche s'est avérée contre-productive : le temps passé à formuler le contexte et à interagir avec l'outil a largement dépassé celui qu'aurait nécessité un diagnostic direct des fichiers de configuration, l'erreur en question étant relativement simple à identifier. L'IA a finalement fourni la correction, mais sans que l'étudiante ait développé la compréhension visée par l'exercice.

Cette observation, bien qu'isolée, illustre une limite inhérente à tout exercice basé sur la soumission de fichiers : la délégation à un outil externe reste possible. Elle souligne également l'importance de l'approche pédagogique active : le véritable apprentissage réside dans le processus de diagnostic, et non dans l'obtention du résultat. Des questions à choix multiples préliminaires ont été intégrées à certains exercices afin de vérifier la compréhension des concepts avant la phase de diagnostic.

5.3 Retours des assistants

Les assistants ayant testé les exercices ont confirmé la pertinence des thématiques abordées et la cohérence des erreurs injectées avec les concepts enseignés. Leurs retours ont également permis d'identifier et de corriger plusieurs problèmes dans les fichiers de configuration et les scripts de correction avant la diffusion aux étudiants.

1. https://github.com/KatharaFramework/kathara-lab-checker/issues/39, https://github.com/KatharaFramework/kathara-lab-checker/issues/33

Chapitre 6

Perspectives

Plusieurs axes d'amélioration ont été identifiés au cours de ce mémoire.

6.1 Mise à l'échelle

L'architecture actuelle repose sur deux agents INGInious, chacun connecté à sa propre machine virtuelle Docker, permettant l'exécution de deux corrections en parallèle. Cette configuration est suffisante pour des groupes de taille réduite, mais pourrait devenir un goulot d'étranglement pour des groupes plus importants soumettant simultanément.

La mise à l'échelle du système est néanmoins envisageable de manière relativement directe : l'ajout d'agents INGInious supplémentaires, chacun associé à une machine virtuelle Docker dédiée, permettrait d'augmenter le débit de correction. Cette extension ne nécessite pas de modification du framework ni des scripts de correction, mais uniquement la mise en place de nouvelles machines virtuelles et leur enregistrement auprès de la plateforme INGInious. Le scheduling round-robin natif distribuerait alors automatiquement les soumissions sur l'ensemble des agents disponibles.

6.2 Mode examen.

L'ensemble du framework repose sur une correction automatique immédiate, le feedback étant transmis à l'étudiant dès la soumission. Certains contextes d'évaluation, tels qu'un examen ou un test chronométré, nécessitent cependant de différer cette correction afin d'éviter qu'un étudiant n'itère sur ses soumissions en exploitant le feedback en temps réel.

Ce mode est réalisable avec l'infrastructure actuelle selon le protocole suivant :

1. Avant l'examen, modifier le script `run` pour désactiver l'exécution de *Kathara Lab Checker*, ou le remplacer par une suite réduite de tests vérifiant uniquement la conformité formelle de l'archive soumise (présence des fichiers attendus, validité du ZIP).
2. À la deadline, rendre la tâche inaccessible via les paramètres INGInious afin d'empêcher toute nouvelle soumission.
3. Restaurer le script `run` original intégrant la correction complète.
4. Relancer automatiquement toutes les soumissions enregistrées en un clic depuis l'interface INGInious, de préférence pendant un créneau creux (la nuit) afin de ne pas saturer la file d'attente INGInious pendant que des étudiants travaillent.
5. Une fois les corrections terminées, rendre la tâche à nouveau accessible aux étudiants en lecture seule afin qu'ils consultent leur feedback, tout en désactivant la possibilité de soumettre à nouveau.

Cette approche ne nécessite aucune modification de l'infrastructure et s'appuie entièrement sur des fonctionnalités natives d'INGInious.

6.3 Automatisation des mesures de temps

Des pistes ont été identifiées pour automatiser ces mesures de manière plus rigoureuse dans des travaux futurs. L'équipe de développement d'INGInious suggère d'utiliser directement le client Python de la plateforme afin d'envoyer plusieurs soumissions simultanées en contournant la limite par étudiant, en s'inspirant du plugin *simple grader* . Une alternative consiste à utiliser l'outil `inginious-task-tester` en le lançant en parallèle pour simuler plusieurs étudiants concurrents. Ces approches dépassent le cadre de ce mémoire mais constitueraient une base solide pour des mesures de charge plus reproductibles.

Chapitre 7

Conclusion

Ce mémoire avait pour objectif de concevoir un framework permettant de créer des laboratoires réseau avec correction automatique, destinés à l'enseignement des réseaux informatiques et de la cybersécurité.

Le chapitre 2 a montré qu'aucun des outils existants ne réunit simultanément l'émulation réaliste, la correction automatique et la distribution d'exercices personnalisés. Ce constat a motivé le choix de Kathará, combiné au *Kathara Lab Checker* et à la plateforme INGInious, comme base technologique du framework.

La solution, développée au chapitre 3, repose sur une approche active de diagnostic : l'étudiant reçoit un réseau préconfiguré contenant des erreurs intentionnelles qu'il doit identifier et corriger. La génération automatique de variants assigne à chaque étudiant une instance unique, garantissant l'équité tout en décourageant le partage de solutions. Plusieurs laboratoires ont été développés et mis en ligne, couvrant des thématiques allant du routage statique à BGP, OSPF et la cybersécurité. Des contributions directes au *Kathara Lab Checker* ont également été réalisées dans ce cadre conduisant à l'ajout de fonctionnalités.

Au chapitre 4, l'architecture déployée a été analysée du point de vue de la sécurité et de la scalabilité. Le mécanisme de scheduling d'INGInious, s'appuyant sur deux agents dédiés chacun connecté à sa propre machine virtuelle, résout le problème de concurrence identifié lors des premiers tests, sans nécessiter de mécanisme de synchronisation.

Le chapitre 5 présente les retours recueillis auprès d'étudiants et d'assistants. Ils ont permis de valider la pertinence de l'approche. L'intérêt exprimé pour l'extension du framework à d'autres cours, notamment en *Computer System Security*, témoigne de son potentiel de réutilisation.

Au chapitre 6, plusieurs axes d'amélioration ont été identifiés pour les travaux futurs, notamment la mise à l'échelle par l'ajout d'agents supplémentaires, l'adaptation à un contexte d'examen et l'automatisation des mesures de charge.

Chapitre 8

Usage de l'intelligence artificielle

J'ai utilisé des outils d'IA (Claude, ChatGPT, Le Chat, GitHub Copilot) **comme assistants** pour certaines tâches, en respectant les principes de l'EPL (**responsabilité**, **authenticité** et **transparence**). Le travail final reste intégralement le mien.

Applications

Écriture

Correction linguistique (orthographe, grammaire, LaTeX) et brainstorming sur mes textes originaux.

Code

Debugging, documentation, refactorisation et implémentation partielle de fonctions *validées et adaptées*.

Engagement

L'IA n'a jamais servi à remplacer ma réflexion, produire du contenu non vérifié ou générer des références non validées.

Annexe A

Scripts

A.1 Script d'assignation des variants

```
const NB_VARIANTS = 144;
const rand = input["@random"][0];
const index = Math.floor(rand * NB_VARIANTS);
const selected = `variant_${index}.zip`;

const url =
  "https://inginious.info.ucl.ac.be/course/tfe-vanhorenbeke/bgp/variations/"
  + selected;

document.write(`<a href="${url}">Download your lab (${selected})</a>`);
```

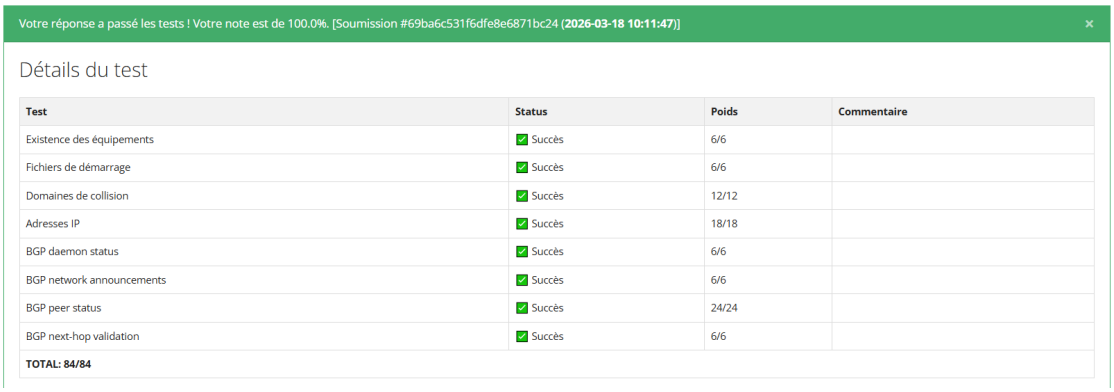
A.2 Récapitulatif généré : `manifest.json`

```json
[
  {
    "variant_id": 0,
    "scheme": "scheme_db8_a",

    "errors": [
      "wrong_local_pref_provider_too_high_as2_to_as4",
      "wrong_local_pref_provider_too_high_as2_to_as1"
    ],

  },
  {
    "variant_id": 1,
    "scheme": "scheme_db8_a",

    "errors": [
      "wrong_local_pref_provider_too_high_as2_to_as4",
      "wrong_local_pref_customer_too_low_as4_to_as7"
    ],

  },


]
```

Annexe B

Exemples de rendus

B.1 Exemple de tableau de feedback



Votre réponse a passé les tests ! Votre note est de 100.0%. [Soumission #69ba6c531f6dfe8e6871bc24 (**2026-03-18 10:11:47**)]

Détails du test

Test	Status	Poids	Commentaire
Existence des équipements	Succès	6/6	
Fichiers de démarrage	Succès	6/6	
Domaines de collision	Succès	12/12	
Adresses IP	Succès	18/18	
BGP daemon status	Succès	6/6	
BGP network announcements	Succès	6/6	
BGP peer status	Succès	24/24	
BGP next-hop validation	Succès	6/6	
TOTAL: 84/84			

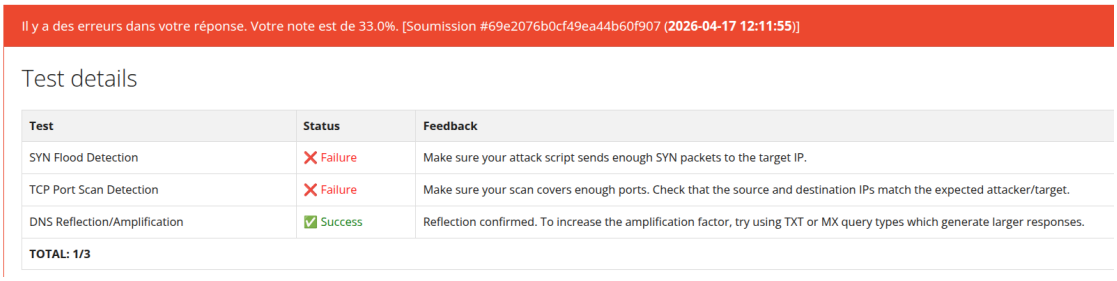
FIGURE B.1 – Soumission réussie

Il y a des erreurs dans votre réponse. Votre note est de 80.0%. [Soumission #69db551b770c95213fbc4f71 (**2026-04-12 10:17:31**)]

Détails du test

Test	Status	Poids	Commentaire
Existence des équipements	Succès	6/6	
Fichiers de démarrage	Succès	6/6	
Domaines de collision	Succès	12/12	
Adresses IP	Succès	18/18	
Status du daemon BGP	Succès	6/6	
Annonces des préfix réseaux BGP	Partiel	4/6	Vérifiez les annonces de réseau BGP dans la configuration (instructions « network » sous « router bgp »)
État des pairs BGP	Partiel	15/24	Vérifier les configurations des pairs BGP : AS distant, adresses IP des voisins, et s'assurer que les pairs sont joignables
Validation du next-hop BGP	Partiel	1/6	Vérifiez l'accessibilité du prochain saut BGP et assurez-vous que les adresses du prochain saut peuvent être résolues
TOTAL: 68/84			

FIGURE B.2 – Soumission partielle



Il y a des erreurs dans votre réponse. Votre note est de 33.0%. [Soumission #69e2076b0cf49ea44b60f907 (**2026-04-17 12:11:55**)]

Test details

Test	Status	Feedback
SYN Flood Detection	❌ Failure	Make sure your attack script sends enough SYN packets to the target IP.
TCP Port Scan Detection	❌ Failure	Make sure your scan covers enough ports. Check that the source and destination IPs match the expected attacker/target.
DNS Reflection/Amplification	✅ Success	Reflection confirmed. To increase the amplification factor, try using TXT or MX query types which generate larger responses.
TOTAL: 1/3		

FIGURE B.3 – Soumission partielle - Labo Cyber

Annexe C

Exercices publiés

Voici la liste des exercices publiés à ce jour, accompagnée d'un bref résumé de leur objectif pédagogique :

- **Routage statique** : configuration simple de quatre routeurs pour introduire les bases du routage IP.
- **BGP** :
  - Simple peering et annonce de préfixes : mise en place d'un voisinage BGP et observation de la propagation des routes.
  - iBGP : échanges internes au sein d'un même système autonome.
  - Relations de peering : compréhension des différents types de relations.
  - Rôles BGP : identification des rôles et comportements dans un réseau BGP.
- **Cybersécurité** :
  - Attaques réseau : implémentation et analyse de trois attaques classiques sur une petite topologie (SYN Flood, TCP Port Scan, DNS Reflection/Amplification).
- **OSPF** :
  - QCM : vérification des connaissances théoriques.
  - Adjacences : étude de l'établissement des voisinages OSPF.
  - Chemin le plus court : calcul de la route demandée.
- **SR-Failures (Segment Routing)** :
  - QCM : compréhension des concepts fondamentaux.
  - IS-IS Fast Reroute : mise en pratique de TI-LFA et SR-MPLS.
- **Anycast** :
  - BGP Anycast avec DoT et DoH : étude du comportement Anycast appliqué aux services DNS sécurisés.

Bibliographie

[1] OpenAI, "GPT-4 Technical Report," *arXiv preprint arXiv :2303.08774*, 2023, Consulté le : 2026-03-05. DOI : 10.48550/arXiv.2303.08774. URL : https://arxiv.org/abs/2303.08774.

[2] Kathará Project, *Kathará Website*, Consulté le : 2026-02-19, 2017. URL : https://www.kathara.org/.

[3] Kathará Project, *Courses Using Kathará*, List of universities using Kathará for teaching, Consulté le : 2026-03-09, 2024. URL : https://www.kathara.org/stories.html.

[4] INGInious Project, *INGInious Automatic Grading Platform*, 2014. URL : https://inginious.org.

[5] P. Schaus, G. Derval et A. Delecluse, "ICLF : An Immersive Code Learning Framework based on Git for Teaching and Evaluating Student Programming Projects," *arXiv preprint arXiv :2601.14814*, 2026. DOI : 10.48550/arXiv.2601.14814. URL : https://arxiv.org/abs/2601.14814.

[6] O. Bonaventure, *Computer Networking : Principles, Protocols and Practice, Fourth Edition*, Consulté le : 2026-04-24, 2025. URL : https://4ed.computer-networking.info/syllabus/default/index.html.

[7] Internet Engineering Task Force, "RFC 2328 — OSPF Version 2," RFC Editor, RFC 2328, 1998. DOI : 10.17487/RFC2328. URL : https://www.rfc-editor.org/rfc/rfc2328.

[8] International Organization for Standardization, "ISO/IEC 10589 — Intermediate System to Intermediate System Intra-Domain Routeing Information Exchange Protocol," ISO, Standard 10589, 2002, Also published as IETF RFC 1142 and RFC7142. DOI : 10.17487/RFC7142. URL : https://www.rfc-editor.org/info/rfc7142/.

[9] Y. Rekhter, T. Li et S. Hares, "A Border Gateway Protocol 4 (BGP-4)," RFC Editor, RFC 4271, 2006. DOI : 10.17487/RFC4271. URL : https://www.rfc-editor.org/rfc/rfc4271.

[10] Docker Inc., *Docker Documentation*, Consulté le : 2026-03-05, 2024. URL : https://docs.docker.com.

[11] FRRouting Project, *FRRouting*, Open-source IP routing protocol suite, Consulté le : 2026-05-22, 2026. URL : https://frrouting.org/.

[12] Internet Systems Consortium, *BIND 9*, Open-source DNS server software, Consulté le : 2026-05-22, 2026. URL : https://www.isc.org/bind/.

[13] lighttpd project, *lighttpd Web Server*, Lightweight open-source web server, Consulté le : 2026-05-22, 2026. URL : https://www.lighttpd.net/.

[14] B. Lantz, B. Heller et N. McKeown, "A Network in a Laptop : Rapid Prototyping for Software-Defined Networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (Hotnets-IX)*, 2010. doi : 10.1145/1868447.1868466. URL : https://dl.acm.org/doi/10.1145/1868447.1868466.

[15] IPMininet Contributors, *IPMininet : IP Routing Protocol Extensions for Mininet*, Developed at UCLouvain, Consulté le : 2026-05-22, 2024. URL : https://ipmininet.readthedocs.io.

[16] M. Jadin, O. Tilmans, M. Mawait et O. Bonaventure, "Educational virtual routing labs with ipmininet," in *ACM SIGCOMM Education Workshop*, 2020, p. 1-5. URL : https://hdl.handle.net/2078.5/227884.

[17] T. Ma et al., "Klonet : an Easy-to-Use and scalable platform for computer networks education," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024, p. 2025-2046. URL : https://www.usenix.org/conference/nsdi24/presentation/ma.

[18] GNS3 Project, *GNS3 Network Emulator*, Consulté le : 2026-03-11, 2024. URL : https://www.gns3.com/.

[19] ns-3 Consortium, *ns-3 Network Simulator*, Consulté le : 2026-03-11, 2024. URL : https://www.nsnam.org/.

[20] Cisco Systems, *Cisco Packet Tracer*, Consulté le : 2026-04-24, 2024. URL : https://www.netacad.com/cisco-packet-tracer.

[21] Containerlab Project, *Containerlab : Container-Based Network Emulation Tool*, Consulté le : 2026-03-11, 2024. URL : https://containerlab.dev/.

[22] S. Marrone, F. Villani et G. Catania, "Kathará : A Lightweight Network Emulation System," in *Proceedings of the 2018 IEEE Symposium on Computers and Communications (ISCC)*, 2018, p. 01 190-01 195. doi : 10.1109/ISCC.2018.8406267. URL : https://ieeexplore.ieee.org/document/8406267.

[23] M. Scazzariello, L. Ariemma et T. Caiazzi, "Kathará : A lightweight network emulation system," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, IEEE, 2020, p. 1-2. DOI : 10.1109/NOMS47738.2020.9110351. URL : https://ieeexplore.ieee.org/document/9110351.

[24] D. Jonassen et W. Hung, "Learning to Troubleshoot : A New Theory-Based Design Architecture," *Educational Psychology Review*, t. 18, p. 77-114, mars 2006. DOI : 10.1007/s10648-006-9001-8.

[25] L. Kaldaras et al., "Employing technology-enhanced feedback and scaffolding to support the development of deep science understanding using computer simulations," *International Journal of STEM Education*, t. 11, juill. 2024. DOI : 10.1186/s40594-024-00490-7.

[26] Van Horenbeke, Noah, *Dépôt GitHub du mémoire*, 2026. URL : https://github.com/NoahVanH/kathara-inginious-lab-framework.

[27] INGInious Project, *Translating Tasks — INGInious Documentation*, Documentation officielle INGInious, Consulté le : 2026-04-24, 2024. URL : https://docs.inginious.org/en/latest/teacher_doc/translating.html.

[28] Docker Inc., *Configure Remote Access for Docker Daemon*, Documentation officielle Docker, Consulté le : 2026-04-24, 2024. URL : https://docs.docker.com/engine/daemon/remote-access/.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**