

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

K8s-LFT: Uma Extensão do LFT para Emulação de Topologias de Rede em Kubernetes

Enzo Z. Celentano

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientador
Prof. Dr. Marcelo Antônio Marotta

Brasília
2025



Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

K8s-LFT: Uma Extensão do LFT para Emulação de Topologias de Rede em Kubernetes

Enzo Z. Celentano

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Prof. Dr. Marcelo Antônio Marotta (Orientador)
CIC/UnB

Prof. Dr. Rodrigo Bonifácio
CIC/UnB

Me. Gustavo Hermínio Araújo
Rede Nacional de Ensino e Pesquisa

Prof. Dr. Guilherme Novaes Ramos
Coordenador do Curso de Engenharia da Computação

Brasília, 08 de setembro de 2025

Dedicatória

Dedico este trabalho à minha família, que sempre me apoiou durante a graduação.

Agradecimentos

Agradeço a todos os professores que contribuíram para a minha formação, em especial ao meu orientador, Dr. Marcelo Marotta.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), por meio do Acesso ao Portal de Periódicos.

Resumo

Este trabalho apresenta o K8s-LFT, uma extensão do *Lightweight Fog Testbed* (LFT) projetada para adicionar ao *backend* baseado em Docker uma arquitetura fundamentada em Kubernetes. A proposta visa modernizar o funcionamento do testbed, incorporando resiliência, observabilidade superiores às disponíveis no LFT original. Para isso, desenvolveu-se uma camada de abstração capaz de selecionar o mecanismo de virtualização desejado, mantendo compatibilidade com experimentos previamente existentes. O sistema foi validado por meio da implementação de uma topologia SDN simples composta por hospedeiros, *switch* Open vSwitch e controlador Ryu, todos executados como pods independentes em um *cluster* MicroK8s. Os resultados demonstram que o uso do Kubernetes possibilita recuperação automática de falhas, maior controle sobre o consumo de recursos e integração nativa com ferramentas de monitoramento como Prometheus e Grafana, permitindo análises detalhadas de CPU, memória e tráfego de rede. A abordagem evidencia o potencial do K8s-LFT como plataforma de experimentação moderna para pesquisas em redes definidas por *Software Defined Network*. Como trabalhos futuros, destacam-se a automação do ciclo experimental, suporte a múltiplos clusters, integração com outros controladores SDN e inclusão de mecanismos avançados de rede.

Palavras-chave: Kubernetes, SDN, emulação de redes, observabilidade, testbed

Abstract

This work presents K8s-LFT, an extension of the *Lightweight Fog Testbed* designed to improve the Docker-based backend with an architecture built on Kubernetes. The proposal aims to modernize the operation of the testbed by incorporating resilience and observability superior to those available in the original LFT. To achieve this, an abstraction layer was developed capable of selecting the desired virtualization mechanism while maintaining compatibility with previously existing experiments. The system was validated through the implementation of a simple SDN topology composed of hosts, an Open vSwitch switch, and a Ryu controller, all executed as independent pods within a MicroK8s cluster. The results demonstrate that the use of this Kubernetes banckend enables automatic failure recovery, finer control over resource consumption, and native integration with monitoring tools such as Prometheus and Grafana, allowing detailed analysis of CPU, memory, and network traffic. The approach highlights the potential of K8s-LFT as a modern experimentation platform for research in software-defined networking. Future work includes full automation of the experimental workflow, support for multiple clusters, integration with additional SDN controllers, and the incorporation of advanced networking mechanisms.

Keywords: Kubernetes, SDN, network emulation, observability, testbed

Sumário

1 Introdução **1**

2 Trabalhos Relacionados **3**

3 Proposta - K8s-LFT **15**

3.1 Fluxo operacional 15

3.2 Arquitetura do K8s-LFT 16

3.2.1 Classe `K8sNode` 17

3.2.2 Classe `K8sHost` 19

3.2.3 Classe `K8sSwitch` 19

3.2.4 Classe `K8sController` 20

3.2.5 Classe `K8sWatcher` 22

3.3 Novas Funcionalidades 23

3.3.1 Observabilidade e Monitoramento 23

3.3.2 Disponibilidade 24

3.4 Visão geral do ciclo 24

4 Resultados **26**

4.1 Definição da Topologia 26

4.2 Seleção do *Backend* 26

4.3 Criação de Instâncias 27

4.4 Inicialização da Topologia 27

4.5 Verificação do Ambiente 28

4.6 Monitoração o Ambiente 28

4.7 Reinicialização do Ambiente 33

4.8 Resumo 33

5 Conclusão e Trabalhos Futuros **34**

5.1 Trabalhos Futuros 35

Referências **36**

Anexo **38**

I driver.py **39**

II SimpleSDNTopology.py **40**

III reset_k8s.sh **42**

Lista de Figuras

3.1 Hierarquia de classes da nova versão proposta do LFT e sua utilização em experimentos de acordo com o motor escolhido. 16

4.1 Listagem dos pods ativos durante a inicialização da topologia. 27
4.2 Configuração da bridge do *switch* `s1` dentro do pod Kubernetes. 28
4.3 Conectividade entre os hospedeiros `h1` e `h2`. 29
4.4 Conectividade entre o hospedeiro `h1` e a *internet*. 29
4.5 Simulação de queda e recriação automática do pod do *switch*. 30
4.6 Interface inicial do Grafana acessada via redirecionamento de porta do microk8s. 31
4.7 Monitoramento do uso de memória dos pods no ambiente K8s-LFT. 31
4.8 Utilização de CPU pelos pods monitorada via Grafana. 32
4.9 Tráfego de rede observado entre os pods da topologia simulada. 32

Lista de Abreviaturas e Siglas

API *Application Programming Interface.*

CPU *Central Processing Unit.*

IaC *Infrastructure-as-Code.*

IP *Internet Protocol.*

K8s Kubernetes.

LFT *Lightweight Fog Testbed.*

LST 2.0 *Lightweight SDN Test-bed 2.0.*

PID Process Identifier.

SDN *Software Defined Network.*

TCP Protocolo de Controle de Transmissão.

TICs Tecnologias de Informação e Comunicação.

UID *Unique Identifier.*

Capítulo 1

Introdução

O desenvolvimento de novas aplicações e protocolos em redes de computadores exige ambientes de teste flexíveis, reprodutíveis e de baixo custo. A construção de experimentos em infraestruturas físicas é frequentemente onerosa, pouco escalável e limitada em termos de controle das variáveis envolvidas, o que impacta negativamente a pesquisa acadêmica e a prototipação de soluções. Nesse cenário, ferramentas de simulação e emulação de redes tornam-se essenciais, pois permitem validar arquiteturas, serviços e políticas em ambientes controlados e altamente personalizáveis [1].

Pesquisas relacionadas a *Software Defined Network* (SDN) reforçam essa demanda, destacando que políticas de segurança, gerenciamento e mecanismos de controle requerem *testbeds* reprodutíveis, configuráveis e extensíveis [2]. Exemplos como o *SDN Testbed* para *cross-layer data-centric security* demonstram a relevância de ambientes experimentais controlados para avaliação rigorosa de comportamentos e políticas [3]. Dentro desse ecossistema de ferramentas, várias soluções foram propostas ao longo dos anos para emulação de redes em diferentes níveis de granularidade e realismo. Entre as mais tradicionais destaca-se o *Mininet* [4], amplamente utilizado no ensino e na pesquisa em SDN. O Mininet permite criar topologias de rede completas, incluindo hosts, switches OpenFlow e links, executando tudo em um único sistema físico por meio de *network namespaces*. Essa abordagem leve e altamente reprodutível consolidou o Mininet como um padrão de fato para prototipação rápida e experimentação de controladores SDN.

Soluções adicionais surgiram para suprir lacunas de orquestração e escalabilidade. O *Containerlab* [5] facilita a criação de laboratórios de rede seguindo princípios de *Infrastructure-as-Code* (IaC), enquanto o OpenConfig-KNE estende o Kubernetes para permitir a criação de topologias arbitrárias em ambientes nativos de nuvem [6]. Em paralelo, ferramentas como o Kathará exploram tanto Docker quanto Kubernetes como mecanismos de virtualização [7]. O *Lightweight SDN Test-bed 2.0* (LST 2.0) [8] reforçou o uso de contêineres para experimentação em SDN, mas ainda sem suporte a um orquestrador moderno. Como

alternativa mais recente, o *Lightweight Fog Testbed* (LFT) [9] foi proposto como um *framework* leve e de rápida prototipação, utilizando contêineres Docker [10] para construir topologias de rede de forma eficiente em termos de recursos e garantir isolamento. Sua arquitetura integra ferramentas como OpenvSwitch e srsRAN 4G, possibilitando a emulação de enlaces com realismo suficiente para experimentos em *Fog*, *Edge Computing* e cenários de segurança. Apesar dessas contribuições, observa-se que ferramentas amplamente utilizadas, em especial o LFT, não incorpora mecanismos modernos de orquestração (como o K8s) que já são utilizados em outras ferramentas [6, 7]. Isso restringe suas capacidades de automação, tolerância a falhas e observabilidade, além da ausência de *frontends* mais acessíveis ao usuário.

A proposta deste trabalho, denominada K8s-LFT [11], diferencia-se por combinar a facilidade de configuração e emulação do LFT original com a orquestração, observabilidade e resiliência proporcionadas pelo Kubernetes. Esse alinhamento com práticas modernas de infraestrutura em nuvem amplia significativamente o conjunto de aplicações possíveis, permitindo a criação de *testbeds* mais robustos, escaláveis e adaptáveis a diferentes cenários de experimentação. Com a expansão do uso do K8s como possibilidade para plataforma de orquestração em ambientes corporativos e acadêmicos, adaptar o LFT para operar sobre Kubernetes representa um passo essencial para modernizar a ferramenta e superar limitações de escalabilidade, automação e recuperação de falhas presentes nas abordagens tradicionais. Assim, este trabalho define os seguintes objetivos específicos:

- adaptar os mecanismos de configuração de rede do LFT para compatibilidade com as abstrações e recursos do Kubernetes, evidenciando as vantagens trazidas por esse motor de orquestração;
- validar a solução proposta por meio da execução de uma topologia de rede representativa em ambiente Kubernetes.

Capítulo 2

Trabalhos Relacionados

O ns-3[12] é um simulador de redes de código aberto, orientado a eventos discretos, amplamente utilizado para pesquisa e ensino em redes de computadores. Ele foi concebido como sucessor do ns-2, com foco em maior modularidade, melhor organização do código e maior fidelidade na modelagem de protocolos e dispositivos de rede. Implementado principalmente em C++, com suporte a bindings em Python, o ns-3 permite a construção de cenários complexos envolvendo redes cabeadas, sem fio, protocolos de transporte, roteamento e aplicações, mantendo controle preciso sobre o comportamento temporal dos eventos simulados.

O simulador é particularmente interessante por buscar um equilíbrio entre abstração e realismo, permitindo que modelos de protocolos se aproximem de implementações reais, inclusive reutilizando código próximo ao utilizado em sistemas operacionais. Além disso, o ns-3 oferece mecanismos avançados de instrumentação, coleta de métricas e reprodução controlada de experimentos, o que favorece a reprodutibilidade científica. Sua ampla adoção pela comunidade acadêmica e a disponibilidade de extensões tornam o ns-3 uma referência consolidada para avaliação de novas propostas em redes.

No que se refere à avaliação experimental, o ns-3 é empregado para analisar desempenho, escalabilidade e comportamento de protocolos e arquiteturas de rede sob diferentes condições. Ele possibilita a medição detalhada de métricas como latência, vazão, perda de pacotes, consumo de energia e utilização de recursos, bem como o estudo do impacto de falhas, mobilidade e variações de carga. Dessa forma, o simulador atua como uma ferramenta fundamental para validar hipóteses antes da implementação em ambientes reais ou de emulação.

A relação entre o ns-3 e o trabalho apresentado está associada ao papel histórico do simulador como alicerce da pesquisa experimental em redes de computadores, especialmente em um período anterior à ampla popularização da virtualização leve e de tecnologias como contêineres. O ns-3 consolidou práticas, métricas e metodologias de avaliação que

influenciaram diretamente a forma como experimentos em redes são concebidos e analisados. Nesse contexto, o K8s-LFT dialoga com esse legado ao adotar uma abordagem complementar baseada em emulação e virtualização com Kubernetes, permitindo que conceitos tradicionalmente avaliados em nível de simulação sejam investigados em ambientes mais próximos da execução real. Dessa forma, o trabalho se posiciona como uma continuidade evolutiva dessa linha de pesquisa, preservando o rigor experimental herdado do ns-3 e incorporando as capacidades modernas de orquestração, controle e observabilidade oferecidas pelo Kubernetes.

O Mininet [4] é uma plataforma de emulação de redes amplamente estabelecida que permite a criação instantânea de protótipos de redes virtuais realistas em uma única máquina. Diferente de simuladores que abstraem completamente o *hardware* e o *software*, o Mininet utiliza virtualização baseada em contêineres leves (através de *namespaces* de rede do Linux e *cgroups*) para instanciar hospedeiros, *switches* e enlaces. Isso permite que cada nó da rede virtual execute o mesmo código de *kernel* e binários de aplicação que seriam executados em uma rede física real, proporcionando um ambiente de alta fidelidade funcional.

A relevância desta ferramenta para a comunidade acadêmica e industrial reside na sua capacidade de oferecer reprodutibilidade e flexibilidade a baixo custo computacional. O Mininet tornou-se o padrão *de facto* para o desenvolvimento inicial de aplicações SDN, pois suporta nativamente o protocolo OpenFlow e permite a conexão com diversos controladores externos. Sua arquitetura permite transitar o código desenvolvido no ambiente emulado para o hardware de produção com pouca ou nenhuma modificação, acelerando o ciclo de desenvolvimento de novas tecnologias de rede.

Em termos de avaliação, o Mininet é utilizado para verificar a correção funcional de topologias de rede e a lógica de controle centralizado. Ele permite medir métricas fundamentais como largura de banda, latência, perda de pacotes e convergência de rotas em cenários controlados. Embora não seja projetado para fidelidade temporal perfeita em altas taxas de transferência (como simuladores de *hardware* dedicados), ele é ideal para validar o comportamento de protocolos e a interação entre o plano de dados (Open vSwitch) e o plano de controle.

No contexto deste trabalho, o Mininet funciona como referência histórica e metodológica para abordagens de emulação baseadas em virtualização leve. Enquanto o Mininet concentra sua operação em um ambiente único e centralizado, o K8s-LFT estende essa lógica para uma infraestrutura distribuída e gerenciada por Kubernetes, permitindo explorar propriedades inexistentes no Mininet, como elasticidade, políticas de reinicialização e integração nativa com ferramentas de monitoramento moderno. Assim, o K8s-LFT segue a mesma linha conceitual, porém adaptado às demandas contemporâneas de resiliência e

observabilidade.

Já o Mininet-WiFi[13] é uma extensão do Mininet voltada à emulação de redes sem fio, com foco especial em cenários de redes definidas por software. A ferramenta permite a criação de topologias que incluem estações móveis, pontos de acesso e enlaces sem fio, mantendo a filosofia do Mininet de executar componentes de rede como processos isolados em um único sistema. O Mininet-WiFi adiciona modelos de mobilidade, propagação de sinal e associação dinâmica, possibilitando a experimentação de redes wireless de forma programável e controlada.

O Mininet-WiFi é particularmente interessante por oferecer um ambiente de emulação que combina simplicidade de uso com um nível elevado de realismo para estudos em redes sem fio. Ele permite a integração direta com controladores SDN, como o OpenFlow, possibilitando o desenvolvimento e teste de aplicações de controle unificado para redes cabeadas e sem fio. Além disso, sua execução em tempo real facilita a depuração de protocolos e aplicações, reduzindo a lacuna entre simulação abstrata e implantação prática.

Experimentalmente, o Mininet-WiFi é utilizado para analisar desempenho, mobilidade, handover, qualidade de serviço e políticas de controle em redes sem fio programáveis. A ferramenta permite observar métricas como latência, perda de pacotes, vazão e impacto da mobilidade dos nós, bem como estudar o comportamento de controladores SDN diante de mudanças dinâmicas na topologia. Dessa forma, ela se mostra adequada para validar propostas de gerenciamento e controle em ambientes wireless complexos.

A relação entre o Mininet-WiFi e o trabalho apresentado está na convergência de objetivos voltados à experimentação flexível e reprodutível em redes definidas por software. Assim como o Mininet-WiFi explora a emulação como alternativa prática à simulação, o K8s-LFT amplia esse conceito ao empregar Kubernetes como base de virtualização e orquestração, oferecendo maior resiliência, escalabilidade e observabilidade. Ambos os trabalhos compartilham a motivação de aproximar os experimentos acadêmicos de ambientes reais, permitindo avaliar soluções SDN em cenários mais próximos da operação prática, porém com controle rigoroso do ambiente experimental.

O *Lightweight Fog Testbed* (LFT)[9] é uma plataforma de código aberto concebida para fornecer um ambiente leve e eficiente para a realização de experimentos e validação de pesquisas em redes e computação em névoa. Em sua arquitetura original, o LFT utiliza contêineres Docker[10] como o principal mecanismo de virtualização, permitindo que os pesquisadores instanciem rapidamente topologias complexas compostas por hospedeiros e elementos de rede virtuais, como o Open vSwitch. Esta abordagem contrasta com os sistemas baseados em máquinas virtuais tradicionais por exigir menos recursos e proporcionar um tempo de *boot* significativamente mais rápido para os nós experimentais.

A principal atratividade do LFT reside na sua capacidade de reprodução de experimentos e na sua interface de gerenciamento simplificada. Ao isolar os componentes da rede em contêineres Docker, o LFT garante que as condições de teste sejam consistentes e facilmente empacotáveis. Ele oferece uma camada de abstração que facilita a configuração e o *deployment* de diferentes arquiteturas de rede, como as SDN, sem a complexidade associada à gerência de *hypervisors* ou sistemas operacionais completos em cada nó, incentivando assim o rápido desenvolvimento de protótipos.

A plataforma LFT é tipicamente utilizada para avaliar a funcionalidade e a lógica de controle de novos protocolos e aplicações, especialmente no contexto de *Fog Computing* e redes definidas por software. Ela surgiu como uma alternativa ao Mininet[4], buscando superar suas limitações em cenários que exigem maior realismo e menor sobrecarga de recursos. Além disso, consegue fornecer métricas básicas de desempenho e consumo de recursos inerentes ao subsistema Docker (*cgroups*).

O LFT é a base direta sobre a qual o K8s-LFT foi construído, mas sua arquitetura originalmente limitada ao Docker restringia sua robustez operacional e seu alcance em termos de monitoramento. Ao incorporar Kubernetes como camada de orquestração, o K8s-LFT preserva parte da filosofia do LFT como leveza, simplicidade e compatibilidade com topologias existentes, ao mesmo tempo em que expande suas capacidades para cenários distribuídos mais complexos. Dessa forma, o sistema proposto evolui o LFT para um ambiente mais alinhado às práticas modernas de experimentação em larga escala.

Kathará[7] é um sistema de emulação de redes leve que permite a construção, execução e análise de topologias complexas utilizando contêineres Linux como unidades básicas de virtualização. Sua arquitetura foi projetada para oferecer simplicidade operacional, portabilidade e baixo consumo de recursos, permitindo que pesquisadores e estudantes executem experimentos de rede em ambientes reais baseados em pilhas de rede completas. O sistema utiliza arquivos de configuração declarativos que descrevem hospedeiros, conexões e serviços, facilitando a reprodução e o compartilhamento de experimentos.

O Kathará é particularmente interessante por combinar precisão na emulação com eficiência na utilização do hardware, além de eliminar a necessidade de configurar manualmente namespaces, roteamento ou interfaces virtuais. Ele fornece uma interface intuitiva e ferramentas auxiliares que tornam a execução de topologias rápida e acessível, mesmo para usuários sem profundo conhecimento em sistemas de virtualização. Outro aspecto relevante é sua independência de plataformas específicas, permitindo seu uso em ambientes acadêmicos, laboratórios de ensino e pesquisas avançadas.

O sistema permite avaliar tanto o comportamento funcional quanto o desempenho de aplicações e protocolos de rede, operando com pilhas completas de TCP/IP e oferecendo suporte à integração de serviços adicionais, como *firewalls*, roteadores e *middleboxes*

personalizados. Com isso, é possível analisar latência, perda, configuração de caminhos, políticas de roteamento e interação entre componentes distribuídos, mantendo alto grau de realismo experimental.

A relação entre Kathará e este trabalho está na visão compartilhada de que a emulação de redes deve ser leve, acessível e baseada em contêineres. Entretanto, enquanto o Kathará otimiza a simplicidade operacional em ambientes locais, o K8s-LFT avança em direção a um ecossistema orientado à orquestração e à resiliência proporcionadas pelo Kubernetes. Assim, o K8s-LFT complementa o espaço ocupado pelo Kathará, posicionando-se como alternativa adequada para experimentos que exigem automação, tolerância a falhas e integração nativa com ferramentas de observabilidade distribuída.

OpenNOP[14] é uma plataforma de observabilidade de redes de código aberto projetada para fornecer visibilidade integrada sobre infraestruturas heterogêneas. Ela unifica a coleta, correlação e análise de métricas provenientes de múltiplos fornecedores e múltiplas camadas da pilha de rede, abrangendo desde o plano físico até serviços e aplicações. A arquitetura do OpenNOP foi construída para operar em ambientes complexos, permitindo que operadores monitorem seu ecossistema por meio de uma visão consolidada e extensível.

A plataforma é notória por sua abordagem modular e por seu foco em interoperabilidade. Ela foi desenvolvido para lidar com ambientes compostos por dispositivos de diferentes fabricantes, superando limitações de soluções proprietárias restritas a ecossistemas específicos. Além disso, a plataforma disponibiliza APIs padronizadas, integração com ferramentas modernas de análise e suporte a pipelines de dados contínuos, o que facilita sua adoção tanto em redes tradicionais quanto em infraestruturas mais dinâmicas.

A plataforma permite avaliar o estado operacional da rede a partir de múltiplas perspectivas, incluindo desempenho, confiabilidade, utilização de recursos e conformidade com políticas. O OpenNOP é capaz de correlacionar dados provenientes de protocolos de telemetria, sistemas de gerenciamento, sondas ativas e métricas de serviços, possibilitando diagnósticos abrangentes. Dessa forma, ele fornece uma base sólida para análises de tráfego, detecção de anomalias, identificação de falhas e compreensão do comportamento da rede em diferentes níveis.

A relação entre o OpenNOP e o presente trabalho emerge do foco comum em observabilidade e na necessidade de fornecer visões integradas sobre o comportamento de sistemas distribuídos. Enquanto o OpenNOP aborda monitoramento em ambientes multi-vendor e multi-layer, o K8s-LFT incorpora mecanismos de observabilidade do ecossistema Kubernetes para experimentos em redes definidas por software. Assim, embora atuem em domínios distintos, ambos compartilham a premissa de fortalecer a visibilidade operacional e fornecer bases analíticas que apoiem a validação de protocolos, diagnósticos e

experimentação em redes modernas.

O piFogBedII[15] é uma plataforma de testes desenvolvida para apoiar experimentos em computação em névoa com suporte à mobilidade, utilizando como base um conjunto de dispositivos físicos de baixo custo, como placas Raspberry Pi. A arquitetura da plataforma permite a criação de cenários realistas envolvendo nós móveis, enlaces variáveis e distribuição dinâmica de serviços na borda da rede. O sistema foi projetado para facilitar a implementação e o estudo de aplicações sensíveis a latência, continuidade de serviço e restrições energéticas típicas de ambientes de *fog computing*.

O piFogBedII proporciona um ambiente experimental que combina fidelidade física, baixo custo operacional e flexibilidade. Ele permite que pesquisadores avaliem diferentes estratégias de alocação de serviços, monitoramento distribuído e gestão de mobilidade em contextos que se aproximam das condições reais da borda da rede. Além disso, sua arquitetura modular permite estender funcionalidades, integrar sensores adicionais e configurar cenários personalizados para diferentes tipos de experimentos.

A plataforma permite avaliar diversos aspectos de sistemas de computação em névoa, incluindo desempenho de comunicação sob mobilidade, impacto de mudanças topológicas dinâmicas, gerenciamento eficiente de recursos limitados e estratégias de migração de serviços. Com isso, o piFogBedII viabiliza análises detalhadas sobre latência, estabilidade, consumo energético, resiliência e continuidade de aplicações distribuídas que dependem de decisões rápidas e da proximidade entre usuários e serviços.

A relação entre o piFogBedII e o presente trabalho surge do objetivo comum de viabilizar experimentação prática em ambientes distribuídos com foco em flexibilidade e reprodutibilidade. Enquanto o piFogBedII utiliza dispositivos físicos para explorar cenários móveis da computação em névoa, o K8s-LFT busca modernizar um testbed de emulação baseado em contêineres ao incorporar Kubernetes como mecanismo de orquestração. Ambos compartilham a motivação de oferecer plataformas acessíveis e configuráveis para validar aplicações distribuídas, ainda que voltadas a domínios distintos: mobilidade física no caso do piFogBedII e virtualização leve e observabilidade ampliada no caso do K8s-LFT.

Containernet[16] é uma extensão do Mininet[4] que substitui os nós emulados tradicionais por contêineres Docker,[10] permitindo a criação de topologias de rede mais realistas e flexíveis. Em vez de utilizar apenas namespaces leves, o Containernet permite que cada host da topologia execute serviços completos, ambientes personalizados e aplicações complexas encapsuladas em contêineres. Dessa forma, a plataforma combina a estrutura do Mininet com a modularidade e portabilidade oferecidas pelo ecossistema Docker.

O Containernet possui como característica principal permitir a inclusão dinâmica de contêineres durante a execução da topologia, algo que não é possível no Mininet con-

vencional. Essa capacidade torna a ferramenta mais adequada para explorar cenários modernos, como redes elásticas, implantação adaptativa de serviços e experimentos envolvendo função de rede virtualizada. Além disso, sua compatibilidade com o ambiente de desenvolvimento Docker facilita a integração com pipelines existentes e acelera o ciclo de desenvolvimento e teste de aplicações distribuídas.

A plataforma é amplamente utilizada para avaliar o comportamento de aplicações e protocolos em topologias flexíveis que combinam rede e serviços. Com suporte a Open vSwitch e controladores SDN, o Containernet permite medir latência, largura de banda, isolamento de recursos e comportamento de aplicações reais executadas dentro dos contêineres. Essa combinação torna a ferramenta útil para estudos de *service chaining*, orquestração de funções, escalonamento dinâmico e experimentos que exigem maior fidelidade do que a oferecida pelo Mininet puro.

A relação entre Containernet e o presente trabalho está na adoção conjunta do paradigma de virtualização leve para experimentação em redes. Enquanto o Containernet aprimora o Mininet ao incorporar contêineres Docker como elementos da topologia, o K8s-LFT segue um caminho complementar ao ampliar essa abordagem com a orquestração nativa do Kubernetes. Assim como o Containernet torna o Mininet[4] mais expressivo e adaptável, o K8s-LFT moderniza o LFT[9] ao adicionar resiliência, observabilidade integrada e mecanismos avançados de gerenciamento, aproximando os experimentos de ambientes reais em nuvem.

O Containerlab[5] é uma ferramenta de código aberto voltada à criação e gerenciamento de laboratórios de redes baseados em contêineres. Seu objetivo é permitir a emulação rápida e reproduzível de topologias de rede compostas por roteadores, *switches* e *hosts* virtuais, utilizando imagens de sistemas de rede reais fornecidas por diferentes fabricantes. As topologias são descritas de forma declarativa, geralmente em arquivos YAML, o que facilita a definição, o versionamento e a recriação de cenários experimentais complexos.

O Containerlab é interessante por reduzir significativamente o esforço necessário para construir ambientes de teste realistas, aproximando a experimentação acadêmica e profissional de cenários operacionais reais. Ao utilizar contêineres leves, a ferramenta oferece inicialização rápida, baixo consumo de recursos e fácil integração com pipelines de automação. Além disso, seu suporte a múltiplos tipos de nós e mecanismos de interconexão torna o Containerlab atrativo tanto para ensino quanto para validação de arquiteturas e protocolos de rede.

Em termos de avaliação, o Containerlab é utilizado para testar comportamento, interoperabilidade e desempenho de dispositivos e protocolos de rede em ambientes controlados. Ele possibilita analisar aspectos como encaminhamento, roteamento, convergência, falhas

e políticas de controle, permitindo a coleta de métricas e a observação do impacto de diferentes configurações topológicas. Dessa forma, atua como uma plataforma de emulação voltada à validação prática de soluções de redes em nível de sistema.

A relação entre o Containerlab e o trabalho apresentado é direta, uma vez que ambos adotam a filosofia de experimentação baseada em contêineres e na definição declarativa de topologias como código, aproximando-se do conceito de infrastructure-as-code. O Containerlab, inclusive citado na introdução, representa o concorrente mais direto dessa abordagem, ao oferecer um ambiente maduro para emulação de redes a partir de descrições declarativas e imagens de dispositivos reais. Nesse sentido, o K8s-LFT dialoga com essa mesma filosofia, porém avança ao integrar Kubernetes como camada de orquestração, explorando mecanismos nativos de resiliência, observabilidade e gerenciamento de recursos em nível de cluster.

O artigo A Hybrid Software and Hardware SDN Simulation Testbed[17] apresenta uma plataforma híbrida destinada à experimentação em redes definidas por *software*. O ambiente combina elementos de simulação em *software* com dispositivos reais, permitindo que pesquisadores avaliem protocolos, algoritmos de controle e configurações de rede em cenários que aproximam o comportamento observado em infraestruturas físicas. A proposta surge da necessidade de equilibrar realismo, flexibilidade e custo computacional, aproveitando técnicas de virtualização e integração com equipamentos de rede.

A principal contribuição do *testbed* está na capacidade de criar ambientes experimentais ricos, sem exigir a construção de laboratórios físicos completos, que são tipicamente caros e difíceis de escalar. A plataforma é interessante porque reduz barreiras de entrada para pesquisas em SDN, oferecendo uma estrutura configurável, reprodutível e capaz de incorporar diferentes componentes reais ou simulados. Essa abordagem torna mais acessível a investigação de comportamentos de rede complexos, permitindo testes iterativos e controlados.

O ambiente descrito no artigo avalia o desempenho de controladores SDN, o comportamento de planos de dados híbridos e a interação entre elementos simulados e reais. Ele possibilita medições relacionadas à latência de controle, taxa de processamento de fluxos, resposta a eventos de falha e impacto de diferentes arquiteturas sobre o encaminhamento de pacotes. Além disso, o testbed analisa como variações na topologia e na composição dos elementos afetam a estabilidade e a eficiência do sistema.

A relação com o K8s-LFT reside no propósito comum de oferecer plataformas modernas e flexíveis para experimentação em SDN. Ambos os trabalhos buscam reduzir custos, aumentar o realismo e facilitar a reprodução de experimentos, convergindo para a construção de *testbeds* eficientes para pesquisa em redes definidas por *software*.

O artigo EdgeNet: Kubernetes as an Edge Computing Testbed[18] descreve o projeto

EdgeNet, que funciona como um testbed global baseado em Kubernetes para experimentação de sistemas distribuídos e de computação de borda. A plataforma utiliza nós voluntários espalhados por diversos provedores e locais, interligados via internet pública, e permite que usuários implantem contêineres em um cluster global multi-tenant. O experimento explora como um ambiente de emulação global pode ser construído usando tecnologia padrão de orquestração de contêineres, evitando grande dependência de hardware dedicado ou de controladores proprietários.

A proposta é interessante porque alinha os benefícios do paradigma de orquestração com os desafios da computação de borda: heterogeneidade de nós, latências variáveis, administração *multi-tenant* e escalabilidade geográfica. O EdgeNet permite que a comunidade de pesquisa tire proveito do ecossistema e da familiaridade com Kubernetes para realizar experimentos em grande escala, distribuídos geograficamente, com menor custo de implantação. O uso de uma infraestrutura compartilhada e baseada em software também acelera o ciclo de experimentação e facilita a reprodução de cenários complexos.

O *testbed* avalia diversos aspectos de ambientes de borda e redes distribuídas, tais como alocação de pods em diversos nós geograficamente espalhados, desempenho sob condições de rede real (com latência e perda), escalabilidade do plano de controle, isolamento entre locatários em um cenário *multi-tenant*, e eficiência de contêineres em ambientes heterogêneos. O estudo também aborda como adaptações em Kubernetes podem tornar possível operar em cenários menos controlados do que data centers tradicionais.

A relação entre esse trabalho e o projeto está no fato de ambos explorarem o uso de Kubernetes para experimentação em ambientes de rede modernos. Enquanto o EdgeNet demonstra como um cluster Kubernetes global pode servir como *testbed* para computação de borda, o K8s-LFT aplica essa filosofia ao domínio da emulação de redes definidas por software, adicionando resiliência, observabilidade e orquestração moderna à plataforma LFT original. Dessa forma, o estudo do EdgeNet fornece um referencial arquitetural para a adoção do Kubernetes em testbeds de rede, complementando a proposta do K8s-LFT para experimentação em SDN com infraestrutura cloud-native.

O FABRIC é um testbed experimental de grande escala voltado à pesquisa em redes avançadas, computação distribuída e sistemas ciberfísicos. Ele fornece uma infraestrutura física real, distribuída geograficamente, composta por enlaces de alta capacidade, recursos computacionais programáveis e dispositivos de rede configuráveis. O objetivo do FABRIC é permitir que pesquisadores conduzam experimentos reprodutíveis e controlados em um ambiente próximo ao da Internet real, integrando recursos de computação, armazenamento e rede de forma unificada.

Esse testbed se torna interessante por oferecer um ambiente experimental aberto e programável, capaz de suportar pesquisas de ponta em áreas como redes definidas por

software, virtualização de funções de rede, edge computing e arquiteturas futuras da Internet. Sua arquitetura orientada a fatiamento de recursos permite que múltiplos experimentos coexistam de forma isolada, explorando hardware real sob controle programático. Além disso, o uso de APIs e ferramentas de automação facilita a criação, o gerenciamento e a repetição de experimentos complexos em larga escala.

Quanto à avaliação, o FABRIC é utilizado para analisar desempenho, escalabilidade, confiabilidade e comportamento de arquiteturas de rede e sistemas distribuídos em condições realistas. Ele possibilita o estudo de métricas como latência fim a fim, vazão, impacto de falhas, elasticidade de recursos e interação entre computação e rede. Dessa forma, o testbed se posiciona como uma plataforma para validação experimental de soluções que exigem maior proximidade com ambientes operacionais reais, indo além das limitações de simulação e emulação local.

A relação entre o FABRIC e o trabalho apresentado está na visão comum de evolução dos ambientes experimentais em redes, partindo de plataformas locais e controladas para infraestruturas mais programáveis e distribuídas. Embora o trabalho não cubra diretamente esse ponto, o FABRIC representa uma evolução natural em direção a ambientes experimentais de larga escala, nos quais conceitos explorados no K8s-LFT poderiam ser estendidos. Nesse sentido, o K8s-LFT pode ser entendido como um passo intermediário entre testbeds baseados em contêineres e ambientes experimentais distribuídos como o FABRIC, preservando controle, reprodutibilidade e observabilidade, ao mesmo tempo em que prepara o caminho para experimentações futuras em infraestruturas reais e federadas.

O artigo *Cloud continuum testbeds and next-generation ICTs: Trends, challenges, and perspectives*[19] apresenta uma revisão abrangente das plataformas de teste (testbeds) voltadas ao conceito de *continuum* de nuvem, ou seja, a integração entre computação em nuvem, névoa (fog) e borda (edge). Os autores categorizam e avaliam testbeds existentes que suportam essa abordagem de contínua interconexão entre diferentes camadas de infraestrutura computacional, destacando que a crescente dependência de tecnologias de informação e comunicação exige sistemas escaláveis, com gestão automatizada e inteligentes.

O estudo é interessante porque oferece uma visão organizada das diversas plataformas de experimento que visam suportar o continuum de nuvem, fornece uma taxonomia prática para caracterizar seus alcances e destaca lacunas ainda presentes em pesquisa e infraestrutura. Além disso, ao focar em ambientes de teste para as próximas gerações de Tecnologias de Informação e Comunicação (TICs), ele evidencia a importância de ambientes experimentais para validar soluções antes que estas entrem em operação, contribuindo para a sustentabilidade do custo-benefício dos sistemas reais.

O artigo avalia os *testbeds* em termos de escopo funcional (por exemplo, suporte a

camadas *edge/fog/cloud*), recursos de automação e gerenciamento, escalabilidade, heterogeneidade de nós, distribuição geográfica e suporte a casos de uso avançados de TICs. Ele também identifica desafios como interoperabilidade, compartilhamento de recursos, latência, segurança, operação multi-tenant e reproducibilidade de experimentos, além de traçar possíveis caminhos futuros para ampliar a eficácia dessas plataformas.

A relação entre esse trabalho e este projeto é o reconhecimento da necessidade de plataformas experimentais modernas e configuráveis para redes e serviços distribuídos. O K8s-LFT, propõe estender o testbed original LFT por meio de orquestração com Kubernetes, visando resiliência, monitoramento e compatibilidade com experimentos existentes. Nesse sentido, os insights do artigo sobre o *continuum* de nuvem ajudam a enquadrar o K8s-LFT no contexto mais amplo de infraestrutura experimental que abrange *edge, fog e cloud*, indicando direções para expansão futura, como heterogeneidade de nós, escalabilidade distribuída e integração entre camadas computacionais.

O artigo de Usman et al. [20] apresenta uma revisão abrangente sobre a observabilidade em infraestruturas distribuídas de *edge computing* e em arquiteturas de microserviços baseadas em contêineres. O trabalho define o conceito de observabilidade, diferenciando-o do monitoramento tradicional, e argumenta que ele é crucial para gerenciar a crescente complexidade de sistemas modernos orquestrados por ferramentas como o Kubernetes. A pesquisa explora como a decomposição de aplicações em microserviços containerizados altera fundamentalmente a maneira como os sistemas devem ser gerenciados e diagnosticados.

A relevância desta pesquisa reside em fornecer o arcabouço teórico para justificar a necessidade de ferramentas de observabilidade em ambientes de contêineres. O artigo detalha os três pilares da observabilidade (*metrics*, *logs* e *traces*) e discute os desafios inerentes ao *troubleshooting* em infraestruturas de borda dinâmicas e fracamente acopladas. Os autores enfatizam que a orquestração por Kubernetes, embora simplifique o *deployment*, também abstrai complexidades que exigem *insights* mais profundos e automatizados para garantir a Qualidade de Serviço (*QoS*).

O artigo avalia as arquiteturas e ferramentas mais utilizadas para implementar a observabilidade. Ele revisa soluções de coleta de métricas e *logs*, incluindo a pilha Prometheus e Grafana, que é o padrão *de facto* para ambientes Kubernetes. A avaliação se estende a como essas ferramentas contribuem para a robustez, manutenibilidade e capacidade de resposta a incidentes. O estudo valida que a análise detalhada de métricas de consumo de recursos é vital para sistemas distribuídos, especialmente em ambientes onde o controle preciso é necessário.

Este *survey* tem uma relação direta com o K8s-LFT, pois fornece uma justificativa acadêmica para a sua principal inovação. A proposta do K8s-LFT de incorporar "resiliên-

cia e observabilidade superiores"pode fornecer testes para uma implementação prática da necessidade que este artigo de revisão demonstra teoricamente. O K8s-LFT utiliza o K8s para automatizar a orquestração e a recuperação de falhas e, em seguida, integra nativamente o Prometheus e o Grafana, o que permite a criação de simulações que satisfazem os requisitos descritos no artigo.

Em suma, a análise dos trabalhos relacionados estabelece que o K8s-LFT se posiciona como um passo evolutivo natural, unificando grande parte das melhores práticas do estado da arte para *testbeds* de rede. Plataformas predecessoras como ns3[12], Mininet[4] e Mininet-WiFi[13], LFT[9], Containernet[16], Containerlab[5] e Kathará[7] provaram a eficácia da emulação leve para SDN, mas são limitadas pela orquestração local (Docker[10] ou *namespaces*) e pela falta de resiliência nativa. Por outro lado, o EdgeNet[18], o *Testbed* Híbrido[17], FABRIC[21] e o piFogBedII[15] demonstram a tendência para infraestruturas distribuídas e o uso de Kubernetes ou *hardware* para *edge computing*. O diferencial do K8s-LFT é justamente preencher a lacuna entre esses dois mundos: ele mantém a fidelidade e a simplicidade da emulação SDN baseada em contêineres, mas integra a resiliência e a orquestração do Kubernetes e incorpora a observabilidade superior via Prometheus[22] e Grafana[23] (conforme justificado por [20] e exemplificado pelo OpenNOP[14]). Assim, o K8s-LFT oferece uma plataforma moderna e completa, adaptada para as demandas contemporâneas de experimentação em redes definidas por *software* com foco em tolerância a falhas e análise detalhada.

Capítulo 3

Proposta - K8s-LFT

Este capítulo apresenta o fluxo operacional, a arquitetura, os componentes e as funcionalidades da solução proposta, denominada K8s-LFT. O código-fonte do projeto [11] encontra-se disponível publicamente, permitindo sua reprodução, extensão e auditoria.

3.1 Fluxo operacional

O projeto foi elaborado para ampliar as capacidades do *framework* original LFT, permitindo que o usuário escolha entre Docker ou Kubernetes como mecanismo de virtualização responsável pela execução da topologia. O processo de criação da topologia inicia-se quando uma requisição de experimento é enviada ao sistema, acionando o módulo central `driver.py`[1], responsável por interpretar os parâmetros fornecidos e definir o *backend* a ser utilizado. Esse módulo é importado por todos os testes de topologia, atuando como ponto único de entrada para a execução dos experimentos, e realiza a importação dos módulos específicos de virtualização. Dessa forma, a escolha realizada no `driver.py` determina diretamente o mecanismo de virtualização empregado por toda a topologia. A partir dessa decisão, a arquitetura garante que todo o fluxo de criação, configuração e inicialização dos nós siga uma lógica semelhante, independentemente do ambiente selecionado. Essa camada de abstração preserva a compatibilidade com a estrutura do LFT original e amplia a capacidade de execução do sistema em ambientes mais observáveis e resilientes, tornando o mecanismo de virtualização um componente substituível do fluxo operacional.

Após a seleção do *backend*, o sistema instancia as classes específicas responsáveis pela criação dos elementos da topologia. Quando a execução ocorre em Docker, são utilizadas as classes `Host`, `Switch` e `Controller`, que reproduzem o funcionamento tradicional do LFT. No contexto de Kubernetes, são acionadas as classes `K8sHost`, `K8sSwitch`

[1] Implementação completa do arquivo no Anexo I

e `K8sController`, que traduzem as operações esperadas pelo emulador para comandos compatíveis com o ecossistema Kubernetes.

A separação modular entre a definição da topologia e o mecanismo de execução assegura que o código do experimento permaneça o mesmo independente do *backend* escolhido. Dessa forma, a maior parte das topologias originalmente desenvolvidas para o LFT continuam funcionando no novo ambiente, sem necessidade de adaptações ou reescrita. Esse desenho arquitetural também facilita futuras extensões, caso outros mecanismos de virtualização ou novos tipos de nós sejam incorporados ao sistema. O resultado é um emulador mais flexível, capaz de acomodar diferentes necessidades de pesquisa e experimentação. Com isso, a arquitetura mantém a simplicidade de uso do LFT original enquanto expande significativamente seu alcance operacional.

3.2 Arquitetura do K8s-LFT

A arquitetura proposta, ilustrada na Figura 3.1, organiza seus componentes por meio de uma hierarquia de classes que define como cada elemento da topologia é criado, configurado e gerenciado. No projeto original da LFT, quando um dispositivo emulado é requisitado pelo experimento, sua criação inicia-se na classe base `Node`, responsável por transformar a descrição abstrata do cenário em entidades concretas dentro do ambiente de virtualização. Essa classe realiza operações fundamentais, como a preparação do ambiente de execução, a configuração das interfaces de rede e a definição das estruturas necessárias para que cada nó possa operar corretamente. Após essa etapa, o nó passa a existir como uma unidade manipulável, permitindo sejam realizadas conexões, ajustes e monitoramento conforme exigido pelo experimento. Assim, a classe `Node` funciona como o alicerce sobre o qual toda a simulação é construída.

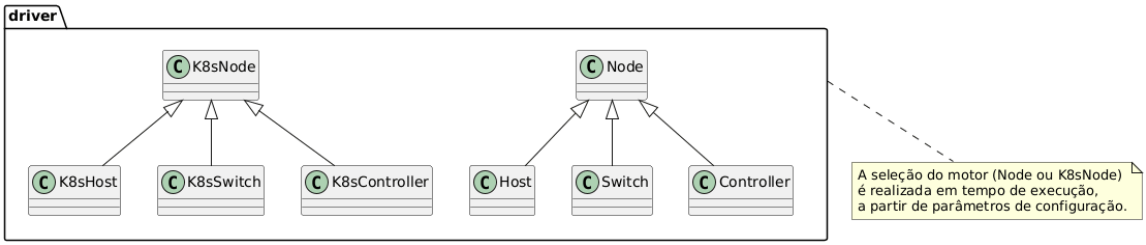


Figura 3.1: Hierarquia de classes da nova versão proposta do LFT e sua utilização em experimentos de acordo com o motor escolhido.

Com o objetivo de preservar a usabilidade e manter a compatibilidade entre as versões do projeto, a classe `K8sNode` foi desenvolvida seguindo o mesmo princípio estrutural de `Node`, mas adaptada ao contexto do Kubernetes. Em vez de criar contêineres diretamente,

essa classe converte as operações do LFT original em comandos executados via `microk8s` e `kubectl`, responsáveis por instanciar e configurar *pods*[2]. Cada pod criado representa um nó da topologia e recebe sua configuração de rede, *namespace* e parâmetros operacionais conforme especificado na requisição inicial. O processo segue até que todos os elementos estejam devidamente implantados por herança, permitindo que o controlador organize os fluxos de comunicação entre hospedeiros e *switches* e conclua a transição entre a requisição inicial e a rede emulada funcional.

A partir dessa hierarquia, surgem as classes especializadas que representam os diferentes tipos de dispositivos da topologia. As classes `Host` e `K8sHost` modelam os nós terminais responsáveis por gerar e receber tráfego, sendo que, no *backend* Kubernetes, cada hospedeiro é executado como um pod baseado na imagem `nicolaka/netshoot` [24]. De maneira semelhante, `Switch` e `K8sSwitch` representam os dispositivos de comutação; no modo Kubernetes, o *switch* é executado em um pod utilizando a imagem `gns3/openvswitch` [25], configurado dinamicamente para compor a topologia definida. Por fim, `Controller` e `K8sController` representam os controladores SDN utilizando a imagem `osrg/ryu` [26], que gerenciam os fluxos de informação entre cada dispositivo.

A organização hierárquica apresentada estabelece, portanto, um fluxo claro: a descrição abstrata da topologia é progressivamente convertida em entidades operacionais capazes de compor a simulação. No modelo original, essa transformação era desempenhada pela classe `Node`; na versão proposta, esse papel fundamental é assumido por sua contraparte no ecossistema Kubernetes.

3.2.1 Classe `K8sNode`

A classe `K8sNode` constitui o núcleo da arquitetura do K8s-LFT, pois grande parte dos componentes, como `K8sHost`, `K8sSwitch` e `K8sController`, herdam diretamente dela. Seu papel é transformar cada nó lógico da topologia em um pod Kubernetes real, utilizando recursos como *StatefulSets*, *namespaces*, capacidades de rede e operações privilegiadas. No momento da construção do objeto, a classe gera um arquivo de kubeconfig, inicializa os clientes da API Kubernetes e registra o nó no sistema de observação (`K8sWatcher`) responsável por acompanhar o estado da topologia. Além disso, define parâmetros fundamentais como imagem, recursos computacionais, contexto de segurança e rótulos que integram o nó ao restante do experimento. Dessa forma, a `K8sNode` estabelece a infraestrutura mínima necessária para que todas as operações posteriores possam ocorrer de modo consistente e reproduzível.

[2]Um pod do Kubernetes é a menor unidade implantável, composta por um ou mais contêineres que compartilham recursos de rede e armazenamento.

O processo de criação do pod é realizado pelo método `instantiate()`, que constrói um manifesto de StatefulSet[3] por meio de `_buildStatefulSetManifest()` e envia a requisição de criação ao *cluster*. Esse manifesto define uma unidade estável com identidade fixa, garantindo que o nome do pod e seu comportamento de rede sejam preservados mesmo em caso de reinicializações ou *crashes* inesperados. Após o envio da requisição, a classe executa `_waitUntilReady()`, que monitora continuamente o pod até que ele entre no estado `Running` e todos os contêineres estejam marcados como prontos. Esse mecanismo assegura que nenhuma operação posterior seja realizada em um nó parcialmente inicializado. Assim, a classe oferece um fluxo robusto para provisionamento de instâncias no cluster.

A `K8sNode` também implementa um conjunto de métodos de rede que permitem manipular diretamente o *namespace* do pod por meio de chamadas ao containerd. O método `_getPodpid()` extrai o Process Identifier (PID) real do contêiner *main* do pod usando o comando `microk8s ctr`, permitindo que interfaces de rede sejam movidas para dentro do *namespace* com comandos como `nsenter`. A partir desse PID, métodos como `connect()`, `_create()`, `_setInterface()` e `setIp()` permitem criar pares *veth*, mover interfaces, configurá-las e atribuir endereços IP de forma manual. Esse nível de controle é essencial para simular cenários de redes definidas pelo usuário, que não dependem da política de rede padrão do Kubernetes. Com isso, a classe concilia os recursos do cluster com o modelo flexível de redes virtuais esperado pelo emulador.

Outra funcionalidade importante é o suporte à persistência de operações, oferecido pelo método `_append_operation()`. Cada ação relevante como conexões, atribuição de IPs ou ajustes de rota é registrada nas anotações do *StatefulSet*, permitindo que o sistema reconstrua o estado da topologia em caso de falhas ou reinicializações. Esse mecanismo funciona como um pequeno *log* transacional dentro do próprio Kubernetes, evitando inconsistências e facilitando a reexecução de cenários experimentais. Dessa maneira, a classe `K8sNode` garante que a simulação permaneça consistente mesmo em ambientes dinâmicos.

Por fim, a implementação também disponibiliza rotinas adicionais para roteamento, como `setDefaultGateway()` e `addRoute()`, que manipulam diretamente a tabela de rotas do pod via `nsenter`. Esses métodos permitem que cada nó atue como um emissor ou receptor de tráfego de acordo com as necessidades do experimento, ampliando o escopo de cenários suportados pelo emulador. A centralidade e generalidade da `K8sNode` tornam possível que quaisquer outros tipos de nós sejam obtidos por simples herança e extensão mínima, mantendo a arquitetura modular e coerente. Sua implementação, portanto, representa um dos elementos mais importantes da adaptação do LFT para Kubernetes.

[3] `https://kubernetes.io/docs/concepts/workloads/controllers/statefulset`

3.2.2 Classe `K8sHost`

A classe `K8sHost` representa os nós terminais da topologia quando o emulador é executado sobre o ambiente Kubernetes, atuando como uma especialização direta da classe `K8sNode`. Nesse contexto, ela não define métodos adicionais, limitando-se a adaptar o conceito de *host* do *framework* original para a criação de um pod baseado na imagem `nicolaka/netshoot`[24]. Toda a lógica de criação, configuração e gerenciamento do pod, incluindo inicialização, associação a *namespaces*, preparação das interfaces de rede e integração com o cluster, é herdada integralmente da `K8sNode`. Dessa forma, a `K8sHost` funciona essencialmente como um identificador funcional dentro da hierarquia, garantindo que o nó seja tratado como um *host* na topologia sem exigir novas rotinas de implementação.

3.2.3 Classe `K8sSwitch`

A classe `K8sSwitch` especializa a `K8sNode` para representar um *switch* virtual baseado em Open vSwitch[4] dentro do cluster Kubernetes. Sua função principal é transformar um nó lógico do tipo *switch* em um pod executando a imagem `gns3/openvswitch`, preparada para operar como elemento de comutação em redes definidas por *software*. No construtor, a classe simplesmente invoca a inicialização da classe-base, herdando todos os mecanismos de criação, monitoramento, registros de operação e manipulação de *namespace*. Com isso, o `K8sSwitch` mantém a mesma infraestrutura de provisionamento da arquitetura, beneficiando-se da robustez da `K8sNode` em operações de rede e integração com o *cluster*. Essa herança direta elimina a necessidade de métodos exclusivos de inicialização além dos específicos à ferramenta Open vSwitch.

O método `instantiate()` estende o comportamento padrão fornecido pela `K8sNode`, primeiro criando o StatefulSet e aguardando sua disponibilidade, e depois ativando um componente essencial: a ponte Open vSwitch. Após o pod estar pronto, o método `_createBridge()` é chamado para verificar se a *bridge* correspondente ao *switch* já existe dentro do contêiner e criá-la caso necessário. Essa criação envolve comandos `ovs-vsctl` executados dentro do pod via `kubectl exec`, garantindo que o elemento de comutação seja configurado corretamente antes de receber quaisquer interfaces. Também é realizada a ativação da ponte com comandos de `ip link`, assegurando que esteja operacional. Dessa forma, o *switch* ingressa na topologia com um plano de controle e dados devidamente inicializado.

[4] O Open vSwitch (OVS) é um comutador virtual multicamada amplamente utilizado em ambientes de virtualização e SDN, oferecendo suporte a OpenFlow, monitoramento avançado e operações de comutação de alto desempenho.

Outro componente crítico da classe é o método `setController()`, responsável por conectar o *switch* ao controlador SDN externo, como o Ryu. Esse método registra a operação no histórico do nó quando apropriado e garante a remoção de controladores antigos antes de aplicar a nova configuração. Em seguida, define o controlador remoto usando os parâmetros de protocolo, endereço e porta e configura o modo de falha como `secure`, impedindo conexões indesejadas com controladores concorrentes. A implementação encapsula a lógica necessária para integrar o *switch* ao plano de controle da topologia SDN. Com isso, o `K8sSwitch` acopla-se ao ambiente SDN de maneira determinística e rastreável.

Para integrar interfaces ao plano de dados, o método `_connectInterface()` associa uma interface de rede à bridge Open vSwitch e ativa o dispositivo. Essa operação é utilizada quando pares *veth* criados pela `K8sNode` são anexados ao *switch*, permitindo conectar *hosts*, outros *switches* e controladores através da ponte. Complementando essa lógica, o método `_createPort()` tenta adicionar uma porta específica ao *switch* correspondente dentro do pod e fornece mensagens de diagnóstico em caso de falha. Essa capacidade de anexação dinâmica de portas permite simular redes arbitrárias de múltiplos saltos dentro do Kubernetes. Assim, o `K8sSwitch` torna-se um elemento fundamental na composição de caminhos e enlaces no experimento.

Por fim, como todos os seus métodos operam sobre estruturas e namespaces fornecidos pela `K8sNode`, o `K8sSwitch` integra-se de maneira completa à arquitetura do K8s-LFT. Ele herda a capacidade de criar veth pairs, mover interfaces para dentro de namespaces de pods e registrar operações relevantes no histórico do experimento, garantindo coerência entre switches, hosts e controladores. Essa integração permite que a classe funcione como componente essencial para ambientes SDN simulados, mantendo a flexibilidade do *framework* original e explorando os mecanismos de isolamento e observabilidade do Kubernetes. Dessa maneira, o `K8sSwitch` cumpre o papel de *bridge* virtual robusta dentro da simulação, habilitando comportamentos complexos de encaminhamento e controle.

3.2.4 Classe `K8sController`

A classe `K8sController` estende a `K8sNode` para representar um controlador SDN executado dentro de um pod Kubernetes. Sua construção especifica a imagem `osrg/ryu`, garantindo que o ambiente possua o ecossistema necessário para inicializar o Ryu, um dos controladores mais utilizados em pesquisas acadêmicas. Como herda todas as funcionalidades de gerenciamento de pods, *namespaces*, interfaces e operações registradas, essa classe concentra-se exclusivamente nos aspectos funcionais relacionados ao plano de controle SDN. Dessa forma, o `K8sController` atua como o nó responsável por gerenciar fluxos e políticas de encaminhamento dentro da topologia simulada. A especialização é mínima, mas cumpre papel essencial por integrar o emulador ao paradigma SDN.

De maneira análoga às outras classes, o método `instantiate()` mantém o comportamento padrão da `K8sNode`, realizando apenas a criação e preparação do pod que hospedará o controlador. Após o pod estar ativo, o componente está pronto para executar o Ryu, embora a classe permita que essa inicialização ocorra posteriormente, conforme necessário pelos experimentos. Essa separação entre criação do pod e inicialização do controlador permite flexibilidade na construção incremental da topologia, principalmente em cenários com múltiplos *switches* conectando-se ao controlador. Assim, o projeto mantém a coerência com o fluxo do *framework* original, no qual controladores são ativados após a estrutura de rede estar configurada.

O método central da classe é `initController()`, responsável por iniciar o Ryu dentro do pod com os parâmetros adequados. Ele registra a operação no histórico do nó, assegurando reprodução futura do experimento, e resolve o endereço IP utilizado pelo controlador, caso não seja explicitamente fornecido. Em seguida, a classe executa o comando `ryu-manager` em segundo plano via `nohup`, especificando a porta OpenFlow e a aplicação Ryu desejada, como `simple_switch_13`. A abordagem permite que o controlador opere de forma contínua, mesmo após múltiplas chamadas subsequentes ou reconexões por parte dos switches. Dessa maneira, a inicialização do controle SDN torna-se totalmente automatizada e replicável.

Para garantir que o controlador esteja operacional, o método privado `__waitForRyu()` monitora periodicamente a porta Protocolo de Controle de Transmissão (TCP) configurada até que ela apareça entre os processos escutando no pod. Esse mecanismo de verificação ativa evita condições de corrida, como *switches* tentarem conectar-se a um controlador ainda não iniciado. Caso o serviço não responda dentro do tempo máximo permitido, um erro explícito é lançado para que o usuário saiba que o controlador não foi configurado adequadamente. Esse comportamento reforça a confiabilidade do emulador, pois impede que a topologia prossiga em estado inconsistente. Assim, a classe provê robustez ao plano de controle.

Por fim, a classe `K8sController` integra-se ao restante da arquitetura ao fornecer um mecanismo consistente para inicializar e manter o plano de controle em funcionamento dentro da topologia simulada. Sua execução do Ryu ocorre de maneira automatizada e monitorada, garantindo que o controlador esteja ativo antes que os switches iniciem suas conexões OpenFlow. Como herda toda a infraestrutura de criação, configuração e persistência já oferecida pela `K8sNode`, o controlador opera de forma homogênea dentro do ecossistema Kubernetes, mantendo comportamento previsível mesmo após falhas ou reinicializações. Essa integração permite que o emulador ofereça um ambiente SDN coerente, no qual o controlador atua como elemento central responsável pela lógica de encaminhamento e pelas decisões de rede. Assim, o `K8sController` consolida-se como o

componente fundamental do plano de controle no K8s-LFT.

3.2.5 Classe `K8sWatcher`

Diferentemente das outras classes do projeto, a classe `K8sWatcher` não representa um componente que participa ativamente no trafego de informações no ambiente tampouco herda de `K8sNode`, atuando somente como o componente responsável por monitorar continuamente o estado dos *pods* envolvidos na simulação, garantindo que a topologia permaneça consistente mesmo diante de reinicializações, falhas transitórias ou recriações automáticas. Implementada como um *singleton*, ela assegura que apenas uma instância seja responsável pela observação centralizada do cluster, evitando conflitos ou tentativas duplicadas de reconfiguração. Seu mecanismo principal consiste em um *watch loop* executado em *thread* dedicada, que consome eventos do servidor de API do Kubernetes e rastreia alterações nos *pods* pertencentes ao experimento. Cada vez que um pod muda de *Unique Identifier* (UID) ou é refeito pelo controlador do *cluster*, o `K8sWatcher` identifica a transição e marca o nó correspondente para reaplicação de operações persistidas. Dessa forma, o sistema mantém a topologia funcional mesmo em ambientes dinâmicos e sujeitos a reinicializações automáticas.

Durante sua execução, o `K8sWatcher` mantém uma estrutura interna que registra o estado de cada pod monitorado, incluindo sua fase atual, número de recriações, UID e um indicador de que operações precisam ser repetidas. A cada evento capturado, ele decide se deve disparar o processo de reconfiguração, distinguindo entre *switches* e demais nós, uma vez que *switches* precisam estar totalmente restaurados antes que *hosts* ou controladores possam se reconectar. Esse comportamento hierárquico evita inconsistências de rede, pois garante que bridges do Open vSwitch sejam recriadas antes de aplicar comandos dependentes, como conexões `veth` ou configurações IP. O método interno `__waitForRunningThenApply()` é responsável por aguardar até que o pod esteja novamente em estado `Running` e, somente então, acionar a reexecução das operações registradas. Com isso, a classe coordena uma restauração ordenada e confiável da topologia.

O processo de reaplicação é realizado por meio do método `reapplyOperations()`, que recupera do StatefulSet as operações previamente registradas pelo nó correspondente. Essas operações são armazenadas como anotações no recurso Kubernetes e representam comandos como `connect()`, `setIp()`, `setDefaultGateway()` ou ações específicas de *switches* e controladores. Após carregar esse histórico, o `K8sWatcher` executa cada operação utilizando o método `executeOperation()`, que identifica o tipo da ação e chama a função apropriada no objeto Python do nó. Essa estratégia permite reconstruir integralmente o estado funcional de cada host, *switch* ou controlador após reinicializa-

ções inesperadas. Assim, a reprodutibilidade da topologia é preservada sem intervenção manual, mesmo em cenários de maior instabilidade.

Por fim, o `K8sWatcher` exerce papel fundamental na resiliência do K8s-LFT ao transformar mudanças de estado do Kubernetes em eventos tratáveis de forma automática. Sua capacidade de detectar erros de conexão com o servidor de API e marcar todos os nós para reconfiguração permite tolerância a falhas temporárias ou congestionamentos na comunicação com o *cluster*. Ao aplicar regras consistentes de restauração priorizando *switches*, sincronizando hosts e preservando dependências o componente garante que a simulação recupere um estado funcional previsível. Dessa forma, o `K8sWatcher` consolida-se como o mecanismo que permite ao K8s-LFT operar de maneira autônoma, robusta e aderente ao comportamento nativo de ambientes Kubernetes distribuídos.

3.3 Novas Funcionalidades

3.3.1 Observabilidade e Monitoramento

Uma das vantagens do *backend* baseado em Kubernetes é a capacidade de integrar, de forma nativa, ferramentas avançadas de observabilidade que são padrão na indústria. No K8s-LFT, essa propriedade é explorada por meio dos *addons* fornecidos pelo `MicroK8s`, que habilitam soluções consolidadas como *Prometheus* [22] e *Grafana* [23]. Essa integração transforma a emulação de topologias de rede em uma plataforma monitorada de forma contínua, com métricas coletadas diretamente da infraestrutura subjacente. A partir desse ecossistema, o usuário passa a acompanhar cada experimento em tempo real, com um nível de visibilidade que não existia no LFT original.

As ferramentas de observabilidade permitem inspecionar, de forma detalhada, o estado operacional dos *pods*, incluindo consumo de CPU, memória, número de reinicializações e disponibilidade geral, além de qualquer outra informação fornecida pelo *Node Exporter*. No contexto de redes, essa visibilidade possibilita avaliar o comportamento da topologia em diferentes cenários, identificar gargalos, analisar picos de tráfego ou detectar anomalias que poderiam comprometer o experimento. O monitoramento contínuo também facilita a comparação sistemática entre diferentes execuções, permitindo que o usuário compreenda o impacto de alterações na configuração, nos enlaces ou no controlador SDN. Assim, a análise deixa de ser empírica e passa a se apoiar em métricas precisas e séries temporais completas.

O acesso às visualizações em tempo real ocorre por meio de um simples `port-forward` para o serviço do Grafana, garantindo que os *dashboards* possam ser consultados sem expor interfaces externas e mantendo o ambiente isolado e seguro. A interface gráfica do Grafana

oferece uma navegação intuitiva por indicadores, gráficos e *time series*, permitindo que a interpretação dos resultados seja fluida mesmo em cenários complexos. Dessa forma, o K8s-LFT eleva o nível de observabilidade da simulação a um patamar comparável ao de clusters reais, consolidando-se como uma ferramenta não apenas para experimentação, mas também para ensino, validação de hipóteses e investigação científica em ambientes SDN.

3.3.2 Disponibilidade

Outra vantagem do *K8s-LFT* é sua capacidade de manter alta disponibilidade durante a execução das simulações. Conforme descrito na seção 3.2.5, essa propriedade é assegurada pela integração entre o Kubernetes e o módulo de monitoramento `K8sWatcher`, implementado no arquivo `watcher.py`. Esse componente acompanha continuamente o estado dos pods, detectando reinicializações, recriações ou quaisquer instabilidades que afetem a execução da topologia. Sempre que um pod retorna ao estado *Running* após uma falha, o `K8sWatcher` aciona automaticamente o processo de reaplicação das operações anteriormente registradas, restaurando sua configuração de rede.

Os pods utilizados na simulação são executados por meio de *StatefulSets*, que garantem identidades persistentes para cada instância. Essa característica permite detectar precisamente quando um componente foi recriado, possibilitando ao sistema restabelecer suas conexões e demais parâmetros de forma automática. Dessa forma, a topologia retorna ao estado operativo original sem necessidade de intervenção manual, preservando a continuidade e a confiabilidade dos experimentos.

3.4 Visão geral do ciclo

O ciclo de execução do K8s-LFT inicia-se com a definição da topologia pelo usuário, por meio de arquivos no diretório `/examples`, como `simpleSDNTopology.py`. Esse arquivo descreve os elementos do experimento (hospedeiros, *switches* e controladores), suas propriedades e os enlaces desejados. Cada classe definida no arquivo de experimento aciona as classes correspondentes em `driver.py`, de acordo com o *backend* selecionado, após determinada se a execução ocorrerá sobre Docker ou sobre Kubernetes.

No caso do backend Docker, a criação das instâncias ocorre por meio de chamadas à classe `Node`, responsável por traduzir as abstrações de `Host`, `Switch` ou `Controller` em operações concretas. O executor solicita o *pull* das imagens necessárias e cria containers individuais para cada elemento da simulação, configurando o ambiente de rede por meio de *bridges* e *namespaces* conforme especificado pelas classes da topologia. O mecanismo de rede nativo Linux, usando *veth pairs* e *nsenter*, estabelece a conexão entre containers e suas

respectivas bridges, enquanto componentes como o Open vSwitch são inicializados dentro dos containers mediante comandos específicos de sua respectiva classe. A inicialização da topologia inclui a ativação dos controladores SDN, a configuração das *bridges* internas e a aplicação das regras de encaminhamento, utilizando a API do Docker para garantir isolamento, monitoração e criação determinística dos elementos envolvidos.

Quando o *backend* selecionado é o Kubernetes, o ciclo de execução assume uma dinâmica distinta. No início do experimento, é instanciado um pod *Watcher*, encarregado de monitorar o estado do cluster por meio de eventos do servidor de API. Em seguida, as chamadas de classes do arquivo de experimento (`Host`, `Switch` ou `Controller`) são convertidas em suas versões equivalentes no ambiente Kubernetes (`K8sHost`, `K8sSwitch` ou `K8sController`), que herdam da classe `K8sNode` para traduzir as operações de alto nível em chamadas para `microk8s kubectl` ou para o cliente da API Kubernetes. Os recursos são então materializados como manifests de *StatefulSets*, aplicados dinamicamente durante a execução. O *kubelet* realiza o *pull* de imagens ausentes e aplica automaticamente configurações de recursos, variáveis de ambiente e *probes* de saúde.

A inicialização da topologia envolve a criação dos pods de *host*, dos *switches* baseados em OVS e de controladores Ryu, além da configuração das *bridges* internas. O estabelecimento de enlaces ocorre de maneira análoga ao cenário Docker, através do mecanismo de rede nativo Linux, usando *veth pairs* e *nsenter*. Outros componentes de observabilidade são habilitados em paralelo, permitindo geração e coleta de métricas durante o experimento. Por fim, também pode-se realizar a limpeza e restauração do ambiente através do script `utils/reset_k8s.sh`, que remove todos os recursos criados e devolve o cluster a um estado inicial e reprodutível para futuras execuções.

Capítulo 4

Resultados

Nesta capítulo, são apresentados os resultados obtidos durante a execução do K8s-LFT, seguindo o ciclo operacional descrito no capítulo 3. Cada etapa é detalhada a partir de um caso prático de uso, demonstrando desde a definição da topologia até a verificação de conectividade e monitoramento do ambiente em execução.

4.1 Definição da Topologia

A topologia de teste é definida pelo usuário em um arquivo Python localizado no diretório `/examples`. Esses programas descrevem os componentes da rede (hospedeiros, *switches* e controladores), bem como suas interconexões. Essa estrutura modular permite ao usuário definir topologias arbitrárias com facilidade, controlando de forma precisa a disposição dos elementos e os parâmetros de cada nó. Com o intuito de exemplificar o funcionamento do projeto, foi escolhida a topologia de rede `simpleSDNTopology.py`.

4.2 Seleção do *Backend*

Conforme previamente citado, o arquivo `driver.py` (código no Anexo I) é responsável por determinar qual *backend* será utilizado e instanciar as classes correspondentes de acordo com a escolha do usuário. Para o experimento apresentado nesta seção, foi adotado o motor Kubernetes, que utiliza a API do `microk8s` para criação e gerenciamento dos pods e redes virtuais.

```
BACKEND = "k8s" # or "docker", for docker-based implementation
```

A lógica acima demonstra a compatibilidade entre as duas implementações, reforçando o caráter extensível do projeto. Assim, a mudança do *backend* requer apenas a alteração de uma única variável, sem impacto sobre o restante do código.

4.3 Criação de Instâncias

Após a seleção do *driver* e do arquivo de topologia, o ambiente de testes pode ser inicializado com o comando[1]:

```
sudo ./<seu_ambiente_virtual>/bin/python3 ./examples/simpleSDNTopology.py
```

O arquivo `simpleSDNTopology.py` (código no Anexo II) segue uma estrutura simples, na qual são instanciados dois hospedeiros, um *switch* e um controlador, todos conectados em uma topologia estrela. Em seguida, são atribuídos endereços IP e *gateways* padrão, e o controlador é inicializado com o *switch* associado.

4.4 Inicialização da Topologia

Após a criação das instâncias, o ambiente inicia o processo de conexão entre hospedeiros, *switches* e controladores. O estado atual dos *pods* pode ser inspecionado diretamente pelo comando:

```
sudo microk8s kubectl get pods -A
```

A Figura 4.1 apresenta o resultado desse comando, evidenciando a criação e o estado de todos os componentes da topologia simulada.

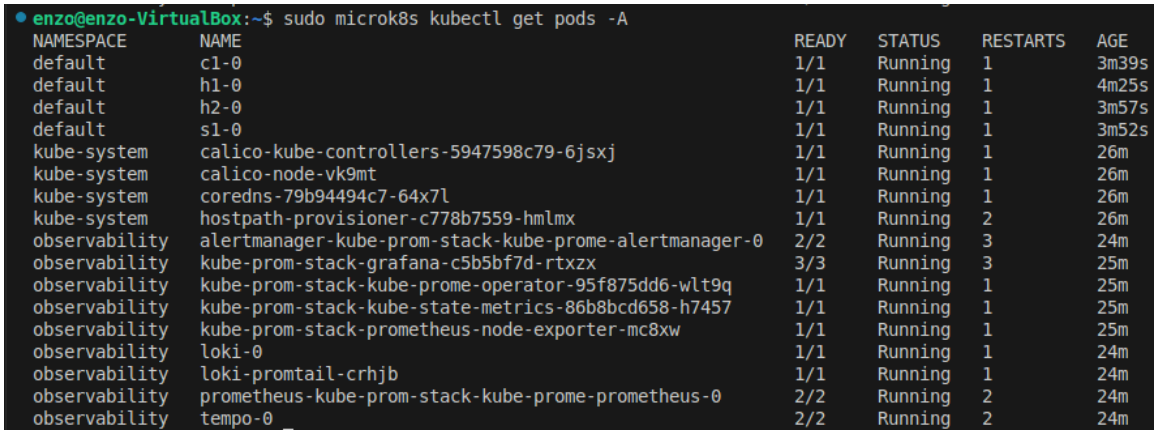


Figura 4.1: Listagem dos pods ativos durante a inicialização da topologia.

[1]Mais informações sobre o processo de inicialização estão disponíveis em `https://github.com/enz03/k8s-lft/blob/main/README.md`

4.5 Verificação do Ambiente

A inspeção individual de cada nó pode ser realizada acessando diretamente o *pod* correspondente, por meio do comando:

```
sudo microk8s kubectl exec -it <nome_do_pod> -- /bin/bash
```

Como exemplo, a Figura 4.2 apresenta a configuração da *bridge* `s1` dentro do pod responsável pelo *switch*.

```
enzo@enzo-VirtualBox:~$ sudo microk8s kubectl exec -it s1-0 -- /bin/bash
s1-0:/$ ovs-vsctl show
35d97b8b-497c-4f87-8ed2-020248ea7b05
    Bridge s1
        Controller "tcp:10.0.0.4:9001"
            is_connected: true
        fail_mode: secure
        Port s1h2
            Interface s1h2
        Port s1c1
            Interface s1c1
        Port s1
            Interface s1
                type: internal
        Port s1host
            Interface s1host
        Port s1h1
            Interface s1h1
```

Figura 4.2: Configuração da bridge do *switch* `s1` dentro do pod Kubernetes.

A conectividade entre os hospedeiros também pode ser verificada com comandos de `ping`, como ilustrado na Figura 4.3, em que observa-se a comunicação bem-sucedida entre `h1` e `h2`. Na Figura 4.4, demonstra-se a comunicação entre um hospedeiro interno e um endereço externo, comprovando a conectividade da topologia com a Internet.

Um dos diferenciais do uso do Kubernetes como backend está na sua capacidade de recuperação automática (*self-healing*). Por exemplo, ao remover o *pod* do *switch* com o comando `sudo microk8s kubectl delete pod s1-0`, o sistema detecta a falha e recria automaticamente o *pod* e suas conexões prévias, conforme ilustrado na Figura 4.5.

4.6 Monitoração o Ambiente

A observabilidade do ambiente é uma das principais vantagens do uso do *Kubernetes* como backend. Para este projeto, foi utilizada a integração nativa do `microk8s` com as

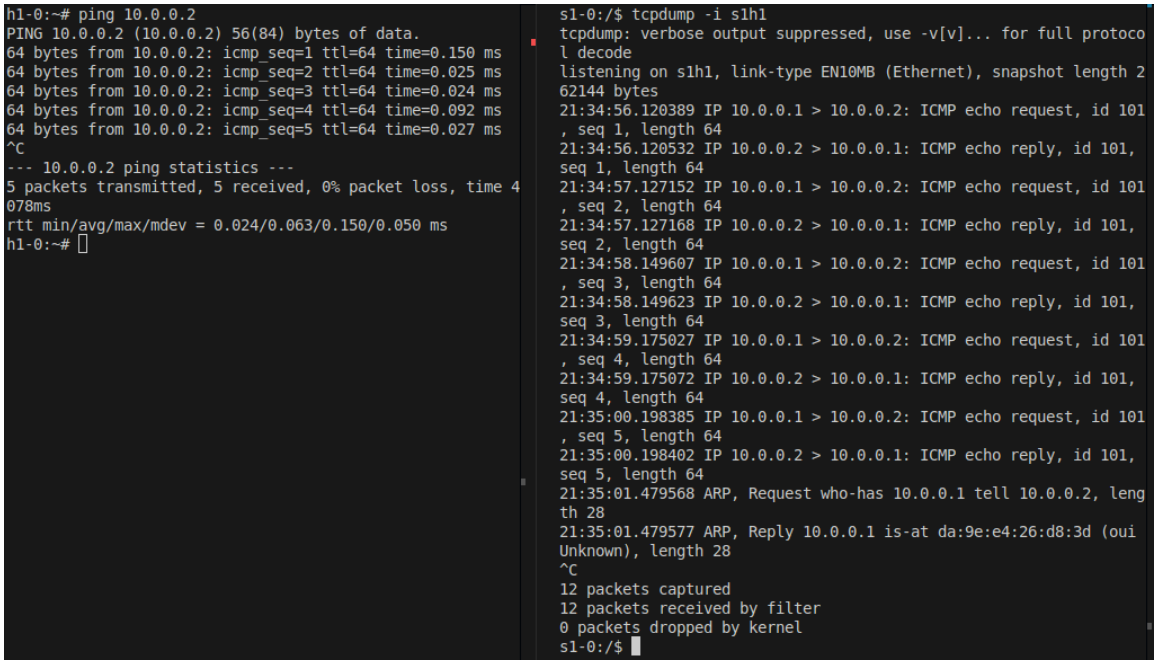


Figura 4.3: Conectividade entre os hospedeiros `h1` e `h2`.

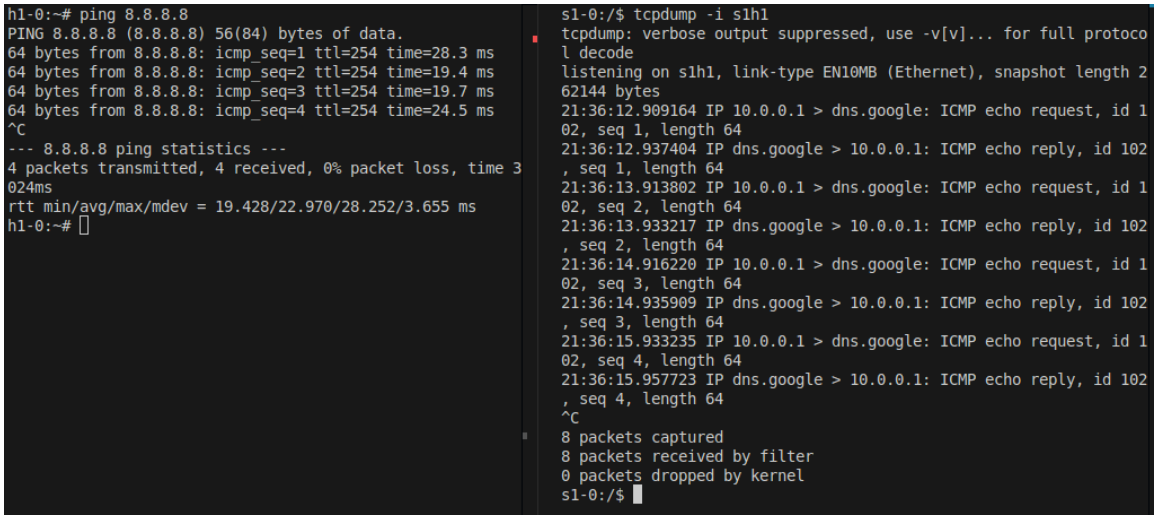


Figura 4.4: Conectividade entre o hospedeiro `h1` e a *internet*.

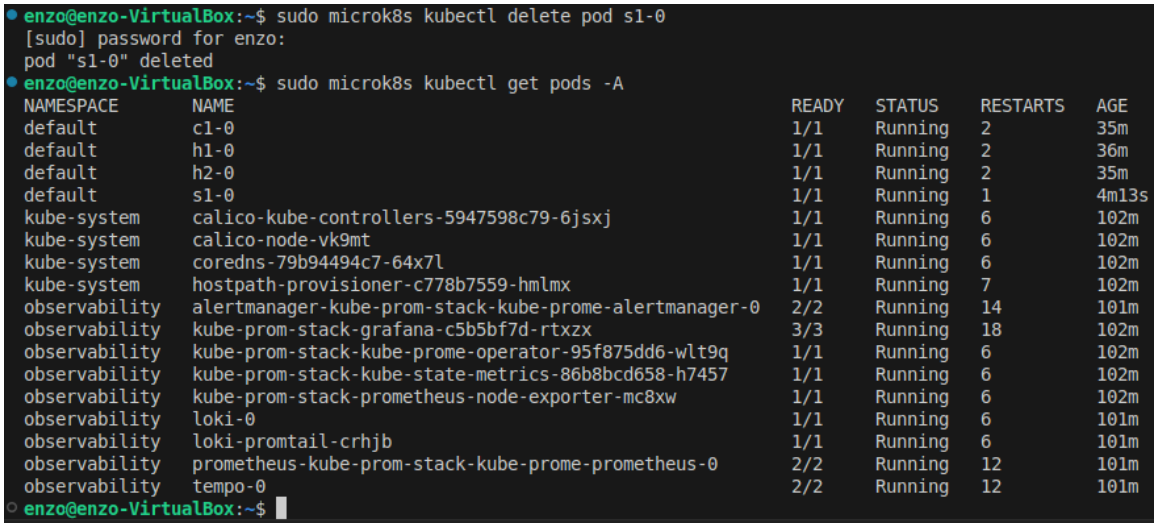


Figura 4.5: Simulação de queda e recriação automática do pod do *switch*.

ferramentas Prometheus e Grafana, que permitem a coleta, armazenamento e visualização de métricas de desempenho dos pods e serviços do cluster em tempo real.

A interface do Grafana pode ser acessada localmente através do redirecionamento de portas, realizado com o comando:

```
sudo microk8s kubectl port-forward -n observability \
svc/kube-prom-stack-grafana 3000:80
```

Após a execução do comando, o painel de controle torna-se acessível via navegador, no endereço `http://localhost:3000`, utilizando as credenciais padrão fornecidas pelo `microk8s` após a instalação do módulo de observabilidade. A Figura 4.6 ilustra a página inicial do Grafana configurado no ambiente de testes.

A partir deste painel, é possível acompanhar o comportamento dos recursos computacionais consumidos pelos pods do K8s-LFT. A título de exemplo, foram escolhidos os recursos de CPU, memória, rede e armazenamento. Esses dados são obtidos a partir das métricas expostas pelo Prometheus e visualizados por meio de gráficos interativos. A Figura 4.7 apresenta o monitoramento do uso de memória pelos pods, permitindo identificar variações de consumo durante a execução das simulações.

De modo análogo, a Figura 4.8 ilustra o gráfico de utilização de CPU, possibilitando observar o comportamento dos processos em cada nó e identificar picos de carga durante a execução dos experimentos.

Por fim, a Figura 4.9 demonstra o uso de rede por pod, representando a taxa de transmissão e recepção de pacotes ao longo do tempo. Essa métrica é particularmente relevante em topologias SDN, nas quais a análise de fluxo de dados é essencial para avaliar o desempenho do controlador e dos *switches*.

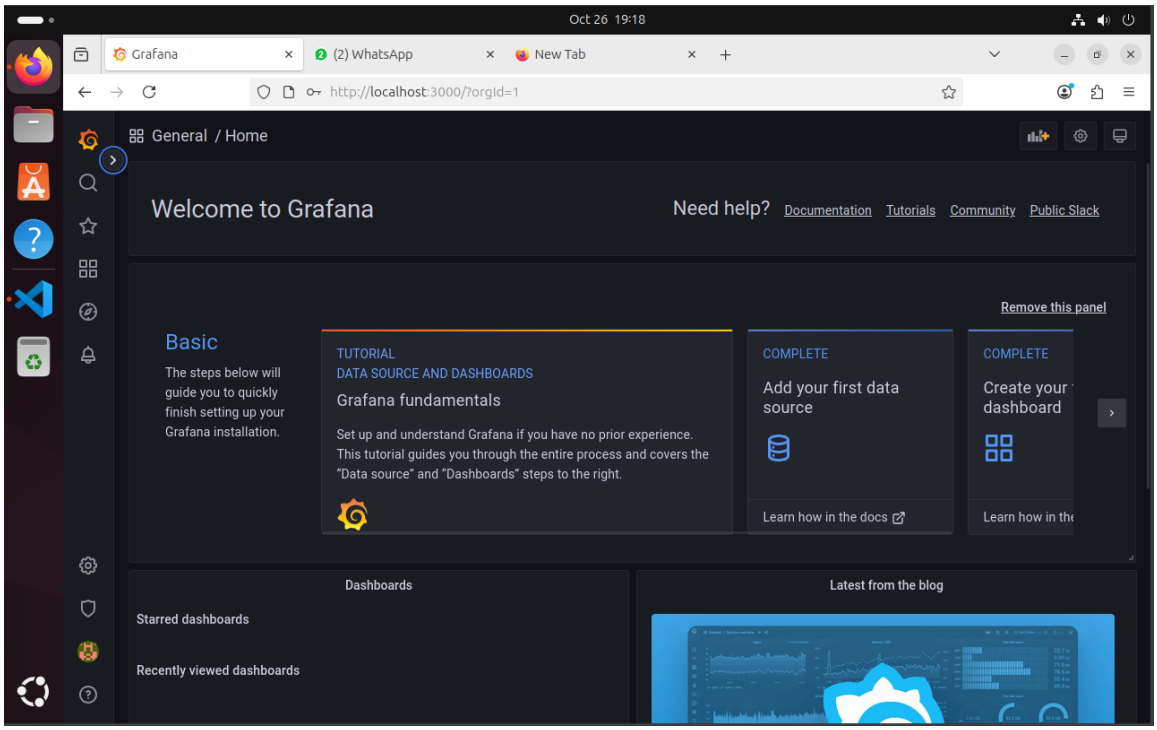


Figura 4.6: Interface inicial do Grafana acessada via redirecionamento de porta do microk8s.

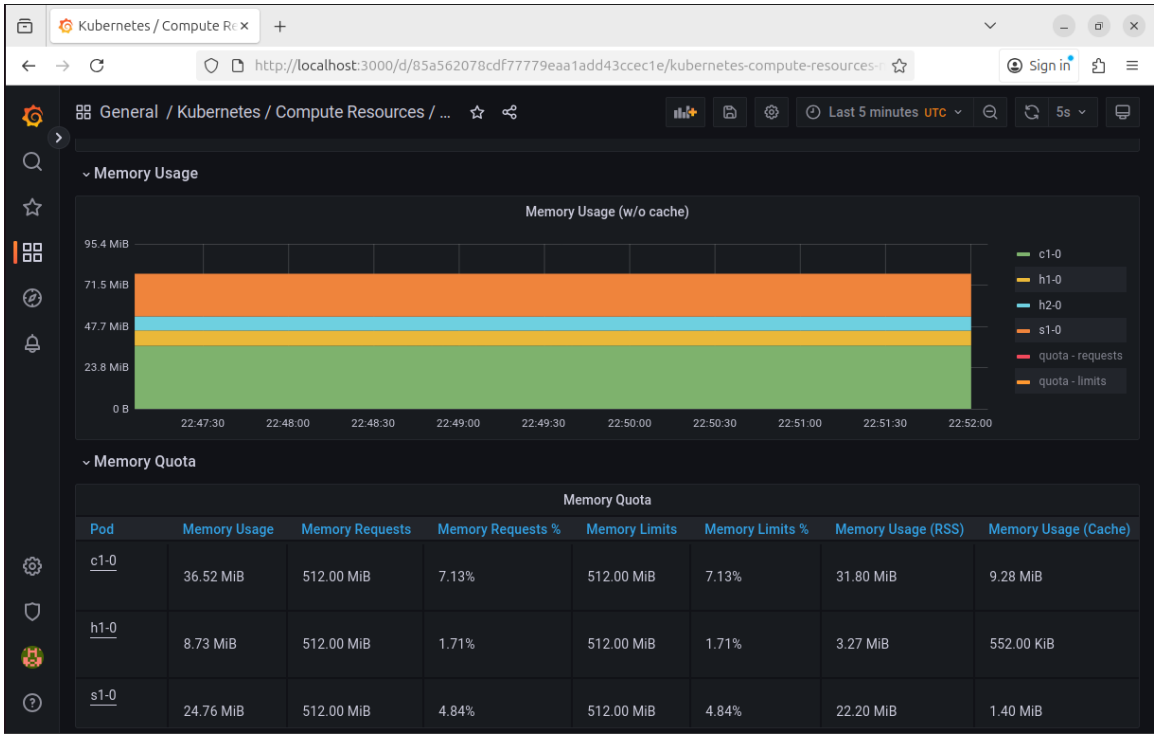


Figura 4.7: Monitoramento do uso de memória dos pods no ambiente K8s-LFT.

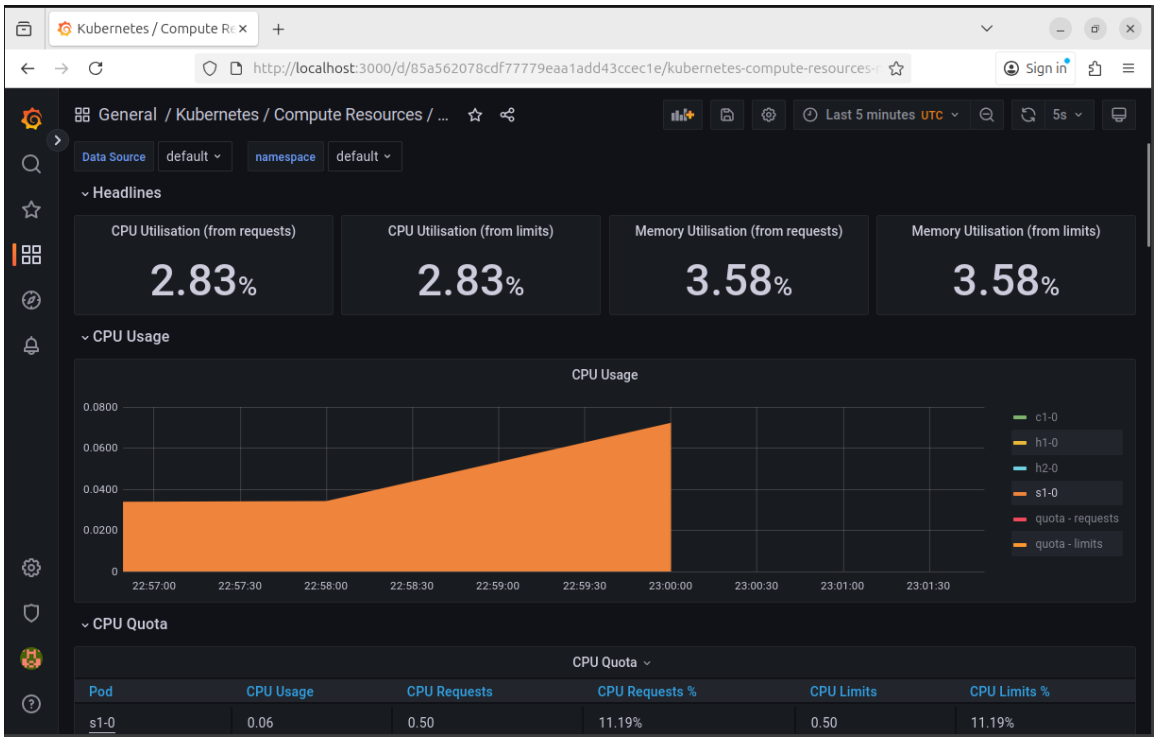


Figura 4.8: Utilização de CPU pelos pods monitorada via Grafana.

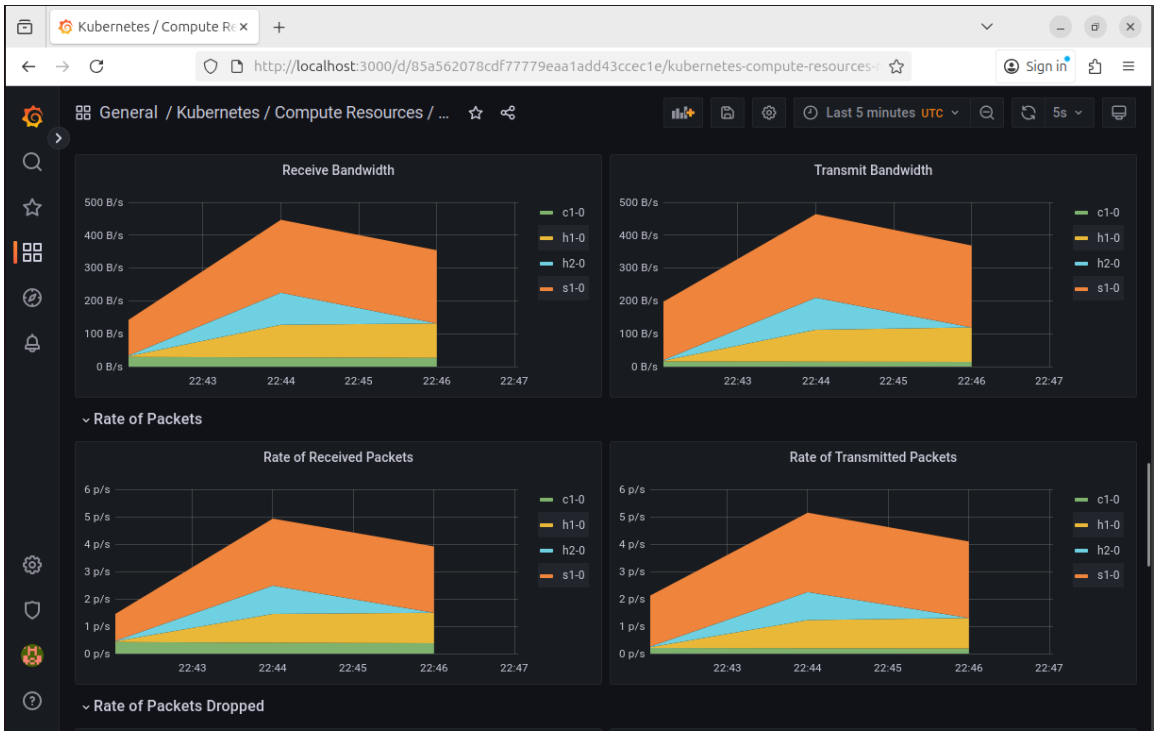


Figura 4.9: Tráfego de rede observado entre os pods da topologia simulada.

Vale ressaltar que além das métricas apresentadas (CPU, memória e rede), o Prometheus coleta automaticamente diversas outras informações sobre o cluster, como latência de operações, uso de disco, disponibilidade de pods, e consumo agregado. Essas métricas adicionais podem ser exploradas por meio de novos painéis personalizados no Grafana, adaptando a visualização conforme o tipo de experimento executado. Em síntese, o uso combinado de Prometheus e Grafana no K8s-LFT oferece um monitoramento abrangente e em tempo real do ambiente de testes, tornando possível a identificação de gargalos, anomalias e padrões de desempenho durante as simulações de rede.

4.7 Reinicialização do Ambiente

O *script* `utils/reset_k8s.sh` (código no Anexo III) permite restaurar o ambiente ao estado inicial. Sua função inclui a exclusão de pods, a limpeza das redes virtuais e a reinicialização dos serviços relacionados. Esse procedimento pode ser utilizado para garantir a consistência entre execuções subsequentes e reduzir o tempo de preparação para novos experimentos.

4.8 Resumo

O ciclo de execução do K8s-LFT mostrou-se eficiente na criação e gerenciamento de topologias tanto no modo Docker quanto no modo Kubernetes. A abordagem baseada em Kubernetes proporcionou maior resiliência e observabilidade, permitindo a realização de experimentos reprodutíveis e o monitoramento detalhado do consumo de recursos e desempenho da rede.

Capítulo 5

Conclusão e Trabalhos Futuros

O desenvolvimento do K8s-LFT teve como principal objetivo modernizar e estender o funcionamento do *framework* LFT, substituindo o *backend* baseado em Docker por uma implementação compatível com o Kubernetes. Essa substituição permitiu a execução de topologias de rede em um ambiente mais robusto, resiliente e observável, mantendo, contudo, a compatibilidade com o modelo de uso e a estrutura modular do projeto original.

A nova arquitetura proposta mostrou-se eficaz na criação, inicialização e monitoramento de topologias SDN em ambiente Kubernetes, permitindo que cada nó da rede (*hosts*, *switches* e controladores) fosse representado por um *pod* independente. Essa abordagem possibilitou simular cenários de falha e recuperação automática, reproduzindo comportamentos realistas observáveis em infraestruturas de produção.

Durante a execução dos experimentos, observou-se que o uso do Kubernetes trouxe ganhos significativos em aspectos como:

- Resiliência: a capacidade de *self-healing* do Kubernetes garante a recriação automática de pods em caso de falha, preservando o estado operacional do ambiente;
- Observabilidade: a integração nativa com *Prometheus* e *Grafana* proporciona visibilidade detalhada do uso de CPU, memória, rede e armazenamento dos componentes da topologia;
- Reprodutibilidade: o uso de scripts e arquivos de configuração versionados assegura que os experimentos possam ser facilmente recriados em diferentes ambientes.

Os resultados apresentados demonstram que o K8s-LFT atinge seu propósito como uma ferramenta de simulação e experimentação para pesquisas em redes definidas por software. Além disso, a compatibilidade com o *backend* original em Docker amplia sua aplicabilidade, permitindo a comparação direta entre diferentes mecanismos de virtualização.

Em síntese, o projeto contribui para o avanço das ferramentas de simulação de redes, fornecendo uma plataforma moderna, extensível e aderente às tecnologias atuais de orquestração de contêineres.

5.1 Trabalhos Futuros

Embora o K8s-LFT tenha alcançado resultados expressivos, diversas oportunidades de aprimoramento e expansão foram identificadas ao longo do desenvolvimento. Entre os trabalhos futuros propostos, destacam-se:

1. Automação completa do ciclo de experimentação: desenvolvimento de uma interface web que permita definir topologias, iniciar execuções e visualizar métricas diretamente, sem necessidade de interação com o terminal;
2. Suporte a múltiplos clusters: extensão do sistema para execução distribuída em mais de um cluster Kubernetes, possibilitando simulações geograficamente distribuídas;
3. Integração com controladores SDN externos: permitir que controladores como ONOS, Floodlight ou Pox sejam instanciados e gerenciados dentro do ambiente do K8s-LFT;
4. Coleta e persistência de métricas históricas: criação de módulos para armazenar dados de desempenho e conectividade em bancos de dados, facilitando a análise longitudinal de experimentos;
5. Incorporação de mecanismos de rede avançados: uso de *Service Mesh*, *Network Policies* e *CNI Plugins* alternativos para enriquecer a simulação e permitir experimentos mais complexos;
6. Integração com ferramentas de Intent-Based Networking (IBN): possibilitando que intenções em linguagem natural sejam traduzidas em topologias e configurações de rede automaticamente;
7. Implementação de um módulo de análise inteligente: uso de aprendizado de máquina para detectar anomalias de desempenho ou prever falhas a partir das métricas coletadas pelo Prometheus.

Com tais melhorias, o K8s-LFT tem potencial para se consolidar como uma plataforma de referência para estudos e experimentações em redes orquestradas e sistemas distribuídos, apoiando tanto o ensino quanto a pesquisa em áreas emergentes da engenharia de redes e da computação em nuvem.

Referências

[1] Braga, Matheus Lorenzato, Ryan da Silva e Jeferson de Limas: *Comparativo de ferramentas de emulação e simulação no ensino de redes de computadores*. Em *Anais do Computer on the Beach*, volume 15, páginas 333–335. Univali, 2024. 1

[2] Kreutz, D. e et al.: *Software-defined networking: A comprehensive survey*. Proceedings of the IEEE, 103, 2015. 1

[3] Wrona, K. e S. Szwaczyc: *Sdn testbed for validation of cross-layer data-centric security policies*. Em *International Conference on Military Communications and Information Systems (ICMCIS)*. IEEE, 2017. 1

[4] Handigol, Nikhil, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz e Nick McKeown: *Reproducible network experiments using Mininet*. Em *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI) – Poster Session*, 2012. `http://mininet.org`, Poster. 1, 4, 6, 8, 9, 14

[5] Nokia: *Containerlab: Infrastructure-as-Code for Networking Labs*. Open Source Project Documentation, 2024. Disponível em: `https://containerlab.dev/`. 1, 9, 14

[6] Foundation, OpenConfig: *OpenConfig-KNE: Kubernetes Network Emulation*. Open Source Project Documentation, 2023. Disponível em: `https://github.com/openconfig/kne`. 1, 2

[7] Scazzariello, Mariano, Lorenzo Ariemma e Tommaso Caiazzi: *Kathará: A lightweight network emulation system*. Em *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, páginas 1–2, 2020. 1, 2, 6, 14

[8] Kaihara, Alexandre, Lucas Bondan, João Gondim, Gabriel Rodrigues, Marcelo Marotta e Genaína Rodrigues: *Lst 2.0: Testbed emulado baseado em contêineres para redes sdn seguras*. Em *Anais Estendidos do XXII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, páginas 111–118, Porto Alegre, RS, Brasil, 2022. SBC. `https://sol.sbc.org.br/index.php/sbseg_estendido/article/view/21699`. 1

[9] Brasília, COMNET Laboratory Universidade de: *Lft: Lightweight fog testbed*. `https://github.com/UnB-COMNET/lft`, 2021. Acesso em: 26 out. 2025. 2, 5, 9, 14

[10] Merkel, Dirk: *Docker: Lightweight linux containers for consistent development and deployment*. Linux Journal, 2014(239):2, 2014. 2, 5, 8, 14

[11] enz03: *k8s-lft: Kubernetes backend for the lightweight fog testbed (lft).* `https://github.com/enz03/k8s-lft`, 2025. Acesso em: 26 out. 2025. 2, 15

[12] Riley, George F. e Thomas R. Henderson: *The ns-3 network simulator.* Em Wehrle, Klaus, Mesut Günes e James Gross (editores): *Modeling and Tools for Network Simulation*, páginas 15–34. Springer, Berlin, Heidelberg, 2010, ISBN 978-3-642-12330-6. 3, 14

[13] Fontes, Ramon L. S. de, Christian E. Rothenberg, Sérgio S. R. Farias, Mauricio M. R. Silva e Marcelo B. Vieira: *Mininet-wifi: Emulating software-defined wireless networks.* 2015 11th International Conference on Network and Service Management (CNSM), páginas 384–389, 2015. 5, 14

[14] Ellsworth, Nathan, Sebastian Troia, Omran Ayoub, Tianliang Zhang e Andrea Fumagalli: *OpenNOP: An open-source network observability platform enabling multi-vendor multi-layer monitoring and ML analysis.* Journal of Optical Communications and Networking, 17(10):D167–D179, 2025. 7, 14

[15] Xu, Qiaozhi, Junxing Zhang e Bulganmaa Togookhuu: *Support mobile fog computing test in piFogBedII.* Sensors, 20(7), 2020. 8, 14

[16] Peuster, Manuel, Holger Karl e Steven van Rossem: *Containernet: A docker-based extension of Mininet*, 2025. `https://github.com/containernet/containernet`, GitHub Repository. See also: Proc. IEEE NFV-SDN 2016. 8, 14

[17] Buzura, Sorin, Adrian Peculea, Bogdan Iancu, Emil Cebuc, Vasile Dadarlat e Rudolf Kovacs: *A hybrid software and hardware SDN simulation testbed.* Sensors, 23(1), 2023. 10, 14

[18] Şenel, Berat Can, M. Mouchet, J. Cappos, T. Friedman *et al.*: *EdgeNet: A global cloud-edge testbed for internet researchers.* IEEE Access, 9:119842–119856, 2021. Often cited as "Kubernetes as an Edge Computing Testbed" in informal contexts. 10, 14

[19] Casino, Fran, Peio López-Iturri e Constantinos Patsakis: *Cloud continuum testbeds and next-generation ICTs: Trends, challenges, and perspectives.* Computer Science Review, 56:100696, 2025. 12

[20] Usman, Muhammad, Simone Ferlin, Anna Brunstrom e Javid Taheri: *A survey on observability of distributed edge & container-based microservices.* IEEE Access, 10:86904–86919, 2022. 13, 14

[21] FABRIC Project: *Fabric: Adaptive programmable research infrastructure for computer science and science applications.* Online, 2021. Available at: https://portal.fabric-testbed.net. 14

[22] Authors, Prometheus: *Prometheus: Monitoring system and time series database.* `https://prometheus.io/`, 2025. Acesso em: 26 out. 2025. 14, 23

[23] Labs, Grafana: *Grafana: Open observability and data visualization platform.* `https://grafana.com/`, 2025. Acesso em: 26 out. 2025. 14, 23

[24] Nicolaka: *Netshoot: a docker and kubernetes network troubleshooting toolkit*. `https://hub.docker.com/r/nicolaka/netshoot`, 2025. Acesso em: 26 out. 2025. 17, 19

[25] GNS3: *Open vswitch docker image*. `https://hub.docker.com/r/gns3/openvswitch`, 2025. Acesso em: 26 out. 2025. 17

[26] Group, OSRG Open Source Research: *Ryu sdn framework docker image*. `https://hub.docker.com/r/osrg/ryu`, 2025. Acesso em: 26 out. 2025. 17

Anexo I

driver.py

```
from profissa_lft.host import Host as DockerHost
from profissa_lft.switch import Switch as DockerSwitch
from profissa_lft.controller import Controller as DockerController

from k8s_lft.host import K8sHost
from k8s_lft.switch import K8sSwitch
from k8s_lft.controller import K8sController

BACKEND = "k8s"  # or "docker", for docker-based implementation
if BACKEND == "k8s":
    Host = K8sHost
    Switch = K8sSwitch
    Controller = K8sController
else:
    Host = DockerHost
    Switch = DockerSwitch
    Controller = DockerController
```

Anexo II

SimpleSDNTopology.py

```
from profissa_lft.driver import Host, Switch, Controller

h1 = Host('h1')
h2 = Host('h2')
s1 = Switch('s1')
c1 = Controller('c1')

h1.instantiate()
h2.instantiate()
s1.instantiate()
c1.instantiate()

h1.connect(s1, "h1s1", "s1h1")
h2.connect(s1, "h2s1", "s1h2")
c1.connect(s1, "c1s1", "s1c1")

h1.setIp('10.0.0.1', 24, "h1s1")
h2.setIp('10.0.0.2', 24, "h2s1")
s1.setIp('10.0.0.3', 24, 's1')
c1.setIp('10.0.0.4', 24, 'c1s1')

c1.initController('10.0.0.4', 9001)
s1.setController('10.0.0.4', 9001)

s1.connectToInternet('10.0.0.5', 24, "s1host", "hosts1")
```

```
h1.setDefaultGateway('10.0.0.5', "h1s1")
h2.setDefaultGateway('10.0.0.5', "h2s1")
s1.setDefaultGateway('10.0.0.5', "s1")
```

Anexo III

reset_k8s.sh

```
#!/bin/bash
# Deleta todos os pods com a label app=k8s-node no namespace default
sudo microk8s kubectl delete pods -l app=k8s-node -n default

# Deleta todos os StatefulSets com a label app=k8s-node no namespace default
sudo microk8s kubectl delete statefulsets -l app=k8s-node -n default

# Deleta todos os Deployments com a label app=k8s-node no namespace default
sudo microk8s kubectl delete deployments -l app=k8s-node -n default

# Deleta todos os Services com a label app=k8s-node no namespace default
sudo microk8s kubectl delete services -l app=k8s-node -n default
```