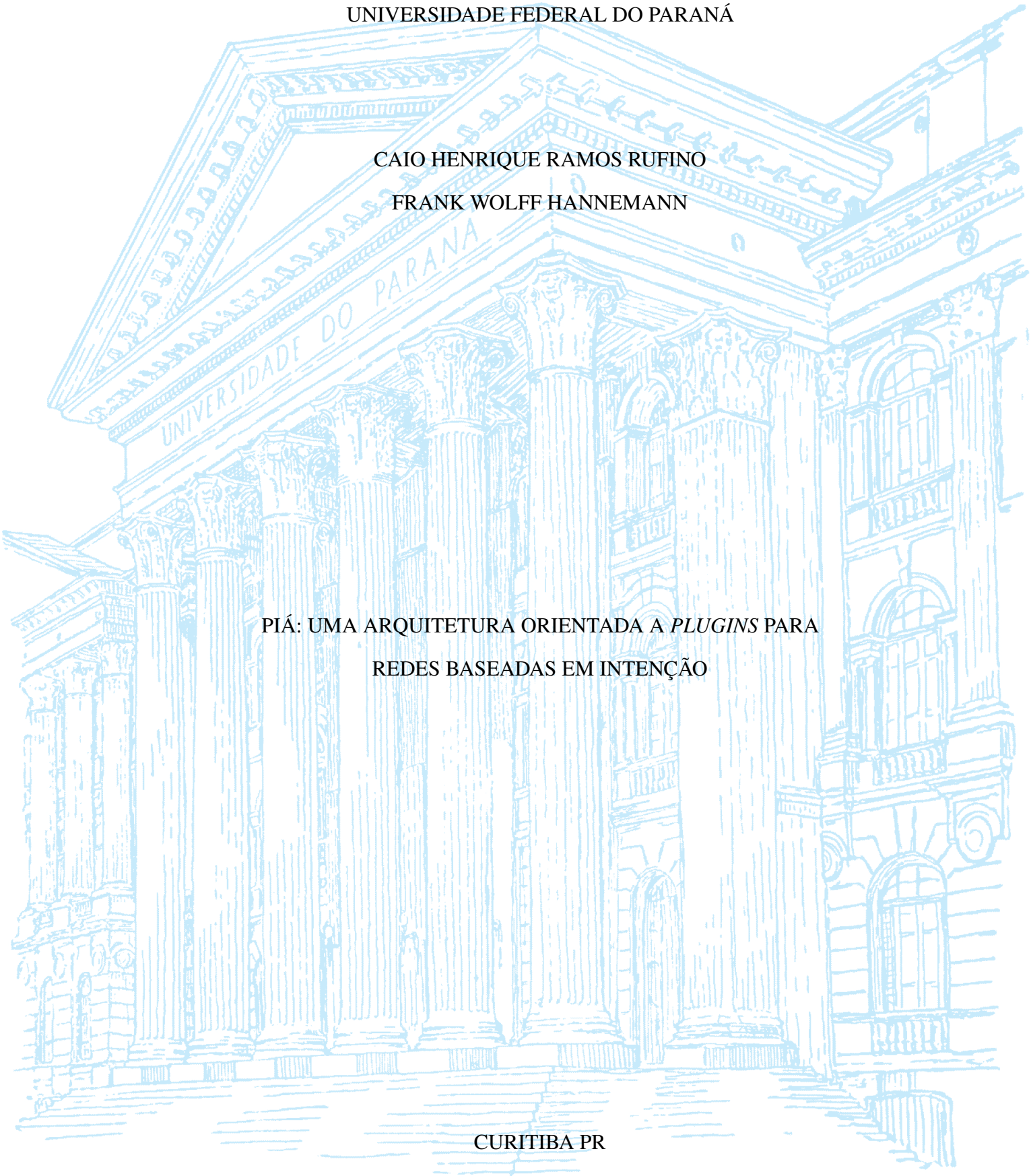
UNIVERSIDADE FEDERAL DO PARANÁ

CAIO HENRIQUE RAMOS RUFINO

FRANK WOLFF HANNEMANN



PIÁ: UMA ARQUITETURA ORIENTADA A *PLUGINS* PARA REDES BASEADAS EM INTENÇÃO

CURITIBA PR

2025

CAIO HENRIQUE RAMOS RUFINO
FRANK WOLFF HANNEMANN

PIÁ: UMA ARQUITETURA ORIENTADA A *PLUGINS* PARA REDES BASEADAS EM INTENÇÃO

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Rede de Computadores*.

Orientador: Giovanni Venâncio de Souza.

CURITIBA PR

2025

Universidade Federal do Paraná
Setor de Ciências Exatas
Curso de Ciência da Computação

Ata de Apresentação de Trabalho de Conclusão de Curso 2

Título do Trabalho: PIÁ: Uma Arquitetura Orientada a Plugins para Redes Baseadas em Intenção

Autor(es):

GRR 20224396 Nome: Caio Henrique Ramos Rufino

GRR 20224758 Nome: Frank Wolff Hannemann

Apresentação: Data: 11 /12 /2025 Hora: 9h30 Local: Auditório Alexandre Direne

Orientador: Giovanni Venâncio de Souza

Membro 1: Luiz Carlos Pessoa Albini

Membro 2: Eduardo Jaques Spinosa

(nome) (assinatura)

AVALIAÇÃO – Produto escrito		ORIENTADOR	MEMBRO 1	MEMBRO 2	MÉDIA
Conteúdo	(00-40)				
Referência Bibliográfica	(00-10)				
Formato	(00-05)				
AVALIAÇÃO – Apresentação Oral					
Domínio do Assunto	(00-15)				
Desenvolvimento do Assunto	(00-05)				
Técnica de Apresentação	(00-03)				
Uso do Tempo	(00-02)				
AVALIAÇÃO – Desenvolvimento					
Nota do Orientador	(00-20)		**********	**********	
NOTA FINAL		**********	**********	**********	100

Os pesos indicados são sugestões.

Conforme decisão do colegiado do curso de Ciência da Computação, a entrega dos documentos comprobatório de trabalho de Conclusão de Curso 2 deve deve respeitar os seguintes procedimentos: o orientador deve abrir um processo no Sistema Eletrônico de Informações (SEI – UFPR); Selecionar o tipo: *Graduação: Trabalho Conclusão de Curso*; informar os interessados: nome do aluno e o nome do orientador; anexar esta ata escaneada e a versão final do PDF da monografia do aluno; Tramitar o processo para CCOMP (Coordenação de Ciência da Computação).

"então, contemplei toda a obra de Deus e vi que o homem não pode compreender a obra que se faz debaixo do sol; por mais que trabalhe o homem para a descobrir, não a entenderá; e, ainda que diga o sábio que a virá a conhecer, nem por isso a poderá achar" Eclesiastes 8:17

AGRADECIMENTOS

Caio: Agradeço, primeiramente, a Deus pela minha existência, pela minha vida e por tudo o que tenho. Agradeço também aos meus pais, que incondicionalmente deram e continuam dando tudo de si por mim e por meus irmãos. Ao professor Giovanni, pela orientação ao longo deste último ano. Aos meus amigos, que, pelos momentos alegres que compartilhamos, tornaram minha vida ainda melhor.

Frank: Agradeço a Deus por seu amor e por ser minha salvação.

À minha mãe, Ligian, por dedicar sua vida para cuidar de mim e do Thomas, por me amar incondicionalmente e ser a melhor mãe do mundo.

À minha família, por sempre me apoiar e me dar todo suporte que eu precisei.

Aos meus amigos, por tornarem a vida mais leve e bem mais divertida.

Ao professor Giovanni, pela parceria e pelas nossas conversas que renderam um trabalho muito bom.

Ao Caio, por essa amizade que rendeu muitas conversas filosóficas e tornaram essa jornada mais legal.

À Milena, minha namorada, melhor amiga e parceira, por me ajudar, me incentivar, me motivar e, principalmente, me amar.

Sem vocês, nada disso seria possível.

RESUMO

A crescente robustez e dinamicidade da Internet tornou as infraestruturas de rede mais complexas, além de dificultar a tarefa de administração e gerenciamento. Na tentativa de simplificar esta tarefa, abstraindo a complexidade do conhecimento técnico exigido por parte dos operadores de rede, surgiu o paradigma IBN (*Intent-Based Network*). O objetivo deste paradigma é transformar intenções em alto nível para configurações de rede de forma automatizada. Apesar do advento da IBN, as arquiteturas existentes possuem casos de uso muito específicos ou são fortemente acopladas a outros sistemas. Deste modo, este trabalho propõe a PIÁ, uma arquitetura genérica, expansível e baseada em *plugins*, que abrange todos os componentes de uma IBN – desde a captura da intenção do usuário até a garantia das intenções. Este trabalho também traz uma implementação da arquitetura proposta. Parte crucial dessa implementação é a PPT (*PIÁ's Parameterized Topology*), um arquivo de texto em formato JSON que busca propor uma padronização, definindo as intenções e configurações da rede. O uso da tecnologia LLM no protótipo possibilitou a tradução de linguagem natural para a PPT, mostrando resultados positivos na tradução da intenção do usuário, o que pode indicar um caminho promissor a ser explorado. Ademais, o módulo de *Assurance* se mostrou eficiente na recuperação de falhas nos experimentos de injeção de erros. É importante ressaltar que este trabalho pode ser visto como um ponto de partida para o desenvolvimento futuro de outras plataformas no contexto de redes IBN.

Palavras-chave: Redes Baseadas em Intenção. Modelos de Linguagem de Larga Escala. Arquiteturas Baseadas em *Plugins*.

ABSTRACT

The growing robustness and dynamism of the Internet has made network infrastructures more complex, as well as making administration and management more difficult. In an attempt to simplify this task, abstracting the complexity of technical knowledge required by network operators, the IBN (Intent-Based Network) paradigm emerged. The goal of this paradigm is to transform high-level intents into network configurations in an automated manner. Despite the advent of IBN, existing architectures have very specific use cases or are heavily coupled to other systems. Thus, this work proposes PIÁ, a generic, expandable, plugin-based architecture that covers all components of an IBN – from capturing user intent to ensuring intentions. This work also presents an implementation of the proposed architecture. A crucial part of this implementation is PPT (PIÁ's Parameterized Topology), a text file in JSON format that seeks to propose a standardization, defining the intentions and configurations of the network. The use of LLM technology in the prototype enabled the translation of natural language into PPT, showing positive results in the translation of user intent, which may indicate a promising path to be explored. Furthermore, the Assurance module proved to be efficient in recovering from failures in error injection experiments. It is important to note that this work can be seen as a starting point for the future development of other platforms in the context of IBN networks.

Keywords: Intent-Based Networks. Large Language Models. Plugin-based Architectures.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

ACL	*Access Control List*
API	*Application Programming Interface*
BERT	*Bidirectional Encoder Representations from Transformers*
CRF	*Conditional Random Field*
DINF	Departamento de Informática
IA	Inteligência Artificial
IBN	*Intent-Based Networking*
IBNS	*Intent-Based Networking System*
IEEE	*Institute of Electrical and Electronics Engineers*
IP	*Internet Protocol*
JSON	*JavaScript Object Notation*
LLM	*Large Language Model*
LSTM	*Long Short-Term Memory*
ML	*Machine Learning*
MLP	*Multi-Layer Perceptron*
NaaS	*Network as a Service*
NER	*Name Entity Recognition*
NFV	*Network Function Virtualization*
NLP	*Natural Language Processing*
ONAP	*Open Network Automation Platform*
PIÁ	*Plugin-based Intent Architecture*
PPGINF	Programa de Pós-Graduação em Informática
PPT	*Piá's Parameterized Topology*
QA	*Question Answering*
QoS	*Quality of Service*
RFC	*Request For Comments*
RNN	*Recurrent Neural Network*
SDN	*Software Defined Network*
SFC	*Service Function Chaining*
SSoT	*Single Source of Truth*
TCP	*Transmission Control Protocol*
TLS	*Transport Layer Security*
UFPR	Universidade Federal do Paraná
VM	*Virtual Machine*
VNF	*Virtualized Network Function*

YAML YAML *Ain't Markup Language*

SUMÁRIO

1 INTRODUÇÃO

A ossificação da Internet pode ser caracterizada pela perda de flexibilidade e evolucionabilidade dos protocolos de rede. Quando um protocolo torna-se um padrão na indústria ou na sociedade (*e.g.,* TCP - *Transmission Control Protocol*), a introdução de novos protocolos, ou até mesmo a evolução de um mesmo protocolo, torna-se uma tarefa complexa (Papastergiou et al., 2017). A perda de flexibilidade em conjunto com o desenvolvimento de paradigmas computacionais causou um aumento de complexidade para os administradores de rede. Em particular, configurações que não são automatizadas passam a se tornar inviáveis devido à quantidade de procedimentos manuais a serem executados por parte do operador, elevando inclusive o nível de conhecimento exigido sobre a infraestrutura subjacente (Kephart e Chess, 2003).

Nesse contexto, surgiram várias soluções para tentar mitigar esses problemas, como as Redes Definidas por Software (*Software Defined Networks* - SDN) (McKeown et al., 2008) e a Virtualização de Funções de Rede (*Network Functions Virtualization* - NFV) (Chiosi et al., 2012). Esses paradigmas permitem modificar a forma como as redes são gerenciadas, trazendo melhorias significativas para a programabilidade e a flexibilidade da infraestrutura. A evolução dessas tecnologias e a necessidade por redes autônomas podem ter indicado o momento adequado para o surgimento das Redes Baseadas em Intenção (*Intent Based Networks* - IBN) (Leivadeas e Falkner, 2023b).

A ideia principal por trás da IBN é permitir que instruções objetivas e de alto nível (*e.g.,* linguagem natural), sejam efetivamente traduzidas operacionalmente em uma rede (Leivadeas e Falkner, 2023b). Consequentemente, o processo de configuração e implantação das redes é facilitado, diminuindo inclusive o conhecimento técnico necessário por parte do operador da rede (Clemm et al., 2022). Em particular, a tecnologia da computação baseada em nuvem (*Cloud Based Computing*) sustenta o paradigma da IBN, já que fornece a infraestrutura que o viabiliza, além de impulsionar esta solução para o mercado de Redes Como um Serviço (NaaS - *Network as a Service)*.

Uma IBN consiste de diversos componentes com funcionalidades bem definidas. Inicialmente, a intenção do usuário, descrita em alto nível, deve ser traduzida em aspectos operacionais da rede. Esses aspectos deverão ser implantados na rede através de uma configuração física (*e.g.,* instalação de regras de fluxo em *switches* OpenFlow) ou virtual (*e.g.,* instanciação de uma Função Virtualizada de Rede (*Virtualized Network Function* - VNF), como um *firewall*). Após a implantação da rede de acordo com as intenções fornecidas, um mecanismo de garantia (*assurance*) deve monitorar o estado da rede e as suas intenções para garantir que elas continuem a ser atendidas, mesmo perante mudanças no estado global da rede (Leivadeas e Falkner, 2023a).

No contexto tecnológico atual, a automação de processos é crescente, principalmente devido ao uso de Inteligência Artificial (IA). Dessa forma, a adoção das IBNs torna-se uma solução atrativa. No entanto, ainda é necessário solucionar algumas lacunas que permeiam esse campo de estudo, principalmente por ser bastante recente – o termo *Intent Based Network* começou a ser utilizado a partir do final de 2016, com artigos surgindo apenas em 2017.

Dentre as principais lacunas identificadas, um grande desafio encontrado no campo da IBN é a falta de padronização, que implica no desenvolvimento de implementações específicas para cada contexto, além de não serem escaláveis (Leivadeas e Falkner, 2023a; Pang et al., 2020; Minhas et al., 2024; Gupta, 2025). O estabelecimento de RFCs (*Request For Comments*) (Clemm et al., 2022; Li et al., 2022) tem sido importante nesse sentido. No entanto, as definições contidas

nestes documentos não suprem a falta de um *framework* genérico e uma implementação em código que sirva como base para outras soluções, permitindo uma adoção ampla do paradigma IBN. As propostas arquiteturais encontradas na literatura são, em alguns casos, excessivamente teóricas e pouco plausíveis de serem implementadas (Njah et al., 2025), altamente acopladas em outros sistemas e *frameworks* (Yu et al., 2024) ou específicas para um único componente, sem abranger a IBN na totalidade de seus componentes (Wang et al., 2023).

Outro ponto a ser explorado no contexto de IBNs é a tradução de intenções em linguagem natural para aspectos operacionais. Muitos dos esforços empregados para resolver este problema foram colocados no treinamento de redes neurais, como MLPs (*Multi-Layer Perceptron*) e RNNs (*Recurrent Neural Network*) (Leivadeas e Falkner, 2023b), ou no uso de NERs (*Name Entity Recognition*) (Jacobs et al., 2018, 2021). Apesar das tentativas, a imensidão do vocabulário com que o usuário pode expressar uma intenção, a falta de *datasets* disponíveis nesse campo e a particularidade de cada contexto de negócio ainda são grandes obstáculos (Zeydan e Turk, 2020).

Diante dos desafios identificados, o presente trabalho propõe a PIÁ (*Plugin-based Intent Architecture*), uma arquitetura genérica e modular baseada em *plugins* dentro do paradigma IBN. Além disso, a arquitetura é proposta no contexto de redes virtualizadas, viabilizando o primeiro passo para uma ampla adoção das IBNs.

Dentro desse contexto, surge a necessidade de uma ferramenta que sirva como base para definir topologias de rede, a fim de que estas sejam, de acordo com as intenções do usuário, implantadas. Assim, foi considerada a utilização do JMP (Santana et al., 2017), um sistema que permite instanciar uma topologia de rede em Mininet com base em um arquivo estruturado em formato JSON (*JavaScript Object Notation*). Essa solução, no entanto, não permitia a definição de certos parâmetros, como Qualidade de Serviço (*Quality of Service* - QoS) e recursos computacionais dos *hosts*. Considerando isto, este trabalho também propõe uma ampliação do JMP, a PPT (*Piá's Parameterized Topology*, permitindo a criação de topologias mais personalizadas.

Por fim, este trabalho também propõe a utilização de um Modelo de Linguagem de Larga Escala (*Large Language Model* - LLM) para a tradução de intenções definidas pelos usuários em linguagem natural para configurações de rede. Particularmente, a API (*Application Programming Interface*) do Gemini (Team et al., 2023) é empregada para transformar uma intenção em um arquivo de configuração intermediária, o mesmo utilizado no sistema mencionado no parágrafo acima, que atua de base para as demais operações da IBN.

Um protótipo foi desenvolvido, implementando a PIÁ e os demais sistemas adjacentes. Este protótipo foi avaliado e resultados experimentais demonstram que a utilização de LLMs para a tradução de intenções é não apenas possível, como também pode ser um método bastante eficaz de realizar esta tarefa. Além disso, ficou evidente a capacidade do protótipo de detectar violações de intenções simples e de recuperar-se delas, o que pavimenta o caminho para soluções mais robustas.

Portanto, dentre as contribuições deste trabalho estão:

- A definição da PIÁ, uma arquitetura genérica e modular baseado em *plugins* para IBN;
- A PPT, uma extensão do JMP, permitindo a definição de múltiplos parâmetros adicionais para instanciar uma rede utilizando o Mininet com base em um arquivo JSON;
- Um módulo que utiliza um LLM para a geração de um arquivo JSON que especifica uma topologia de rede com intenções a partir de linguagem natural.

O trabalho está organizado da seguinte forma. O Capítulo 2 apresenta a fundamentação teórica necessária para o trabalho, apresentando os principais conceitos relacionados à IBN. O

Capítulo 3 apresenta alguns trabalhos relacionados, suas contribuições no contexto de IBN e as lacunas presentes. No Capítulo 4 é descrita a proposta do trabalho, bem como o protótipo implementado. Já no Capítulo 5 apresentam-se os experimentos realizados. Finalmente, o Capítulo 6 conclui o trabalho e apresenta possíveis trabalhos futuros.

2 FUNDAMENTAÇÃO

Este capítulo busca apresentar a fundamentação teórica que serve como base para este trabalho. Na Seção 2.1 contextualizamos a história dos antecessores da IBN. Na Seção 2.2 expomos os principais conceitos relacionados à IBN, junto de uma explicação sobre os elementos que a constituem.

2.1 HISTÓRICO

Um dos grandes desafios da Internet nas últimas décadas é resolver de maneira suficiente o problema conhecido como ossificação da Internet. Esse problema é caracterizado pela perda de flexibilidade e evolucionabilidade dos protocolos de rede. Para um protocolo ser amplamente utilizado, é provável que exista uma demanda que suporte o protocolo. No entanto, a motivação para usar um protocolo que não atingiu uso em larga escala é praticamente nula. Isso cria um ciclo vicioso que torna a introdução de novos protocolos, ou até mesmo a evolução de um mesmo protocolo, quase impossível. Isso acontece, em grande parte, pela configuração das *middleboxes* (Papastergiou et al., 2017).

As *middleboxes* são dispositivos intermediários que processam o tráfego de rede entre o cliente e o servidor. Muitas delas estão configuradas para somente transmitir dados de formatos conhecidos, descartando os dados desconhecidos, ou seja, qualquer protocolo que não seja padrão é descartado (Edeline e Donnet, 2019). Uma *middlebox* precisa suportar protocolos que são amplamente utilizados no momento de sua implantação, mas não precisam suportar novos protocolos ou extensões de protocolos já existentes. Isso, por sua vez, impede que protocolos ganhem notoriedade suficiente para serem considerados relevantes. Dessa forma, novas *middleboxes* não terão suporte a elas.

Um exemplo de protocolo que sofreu de ossificação é o *Transport Layer Security* (TLS). A versão 1.3 do TLS se provou não implantável por causa das implementações errôneas do protocolo nas *middleboxes*. Como resultado disso, a versão 1.3 do TLS imita a versão 1.2 para poder funcionar. Para isso, ela manda a primeira mensagem (o *client hello*) imitando a versão 1.2 e utiliza outras extensões do protocolo para o seu funcionamento (Sullivan, 2017).

No contexto de uma necessidade crescente de robustez e dinamicidade da Internet, além da perda de flexibilidade que a ossificação trouxe, surgiu o paradigma SDN (*Software-Defined Networking*) (Kulkarni, 2017). O paradigma SDN busca separar a lógica de controle da rede (*i.e.,* plano de controle) dos roteadores e *switches* que encaminham o tráfego (*i.e.,* plano de dados) (Raghavan et al., 2012). Toda a lógica de controle é implementada por um controlador geralmente centralizado, tornando os *switches* de rede simples dispositivos de encaminhamento. Consequentemente, SDN permite uma configuração alto nível da rede, além de simplificar a sua operação uma vez que possui menor dependência de comandos baixo nível e *vendor-specific* (Kreutz et al., 2015).

Outro paradigma que surgiu para transformar a Internet ossificada em um sistema mais ágil, flexível e dinâmico, foi o NFV (Faraci e Schembra, 2015). O paradigma NFV desacopla funções de rede de *middleboxes* dedicados e as transforma em entidades virtualizadas baseadas em software, chamadas VNFs (*Virtualized Network Funcitions*) (Leivadeas e Falkner, 2023b). Isso reduz a dependência de dispositivos físicos dedicados e especializados para os provedores de serviços, alocando e utilizando os recursos virtuais e físicos somente quando e onde for necessário (Chiosi et al., 2012; Mijumbi et al., 2016).

A evolução de tecnologias de rede como SDN e NFV, além da incorporação da inteligência artificial em quase todos os aspectos da sociedade, podem ter sinalizado o momento maduro para o advento das Redes Baseadas em Intenção (*Intent-Based Networking* - IBN) (Leivadeas e Falkner, 2023b).

2.2 *INTENT-BASED NETWORKING*

O surgimento do *Intent-Based Networking* está diretamente ligado à crescente complexidade das redes modernas. À medida em que serviços de rede se tornam mais sofisticados, a sua configuração e gerenciamento também tendem a se tornar mais desafiadores. Cada fabricante de equipamentos de rede (*e.g.,* Cisco, Juniper, MikroTik, entre outros) adota sua própria "linguagem" de configuração, com sintaxes e comandos distintos. Isso exige que os administradores de rede conheçam múltiplas formas de realizar tarefas semelhantes, impossibilitando novos usuários com menos conhecimento técnico de fazerem qualquer alteração na configuração da rede.

Essa crescente complexidade impulsionou o surgimento de novos paradigmas de oferta e gestão de redes, com as Redes Como um Serviço (NaaS). NaaS é um *framework* de serviços que integra oferta de computação em nuvem com acesso direto e seguro do locatário à infraestrutura de rede (Costa et al., 2012). Em particular, as redes IBN apresentam grande potencial para simplificar a experiência dos usuários nesse tipo de plataforma, uma vez que diminuem o conhecimento técnico exigido por parte dos operadores da rede para realizar a sua configuração, implantação e gerenciamento (Toy, 2021).

Além disso, a difusão da computação em nuvem nos últimos anos acarretou a adoção em larga escala de ambientes *multi-cloud*, em que organizações utilizam serviços de várias provedoras de *cloud* para satisfazerem suas necessidades (Imran et al., 2020). No entanto, para gerenciar esses ambientes que exigem performance e segurança, a IBN se destaca (Singh, 2025).

A IBN é uma rede que possui uma camada de abstração no processo de seu gerenciamento, permitindo que o usuário defina através de uma intenção o "quê" a rede deve fazer, deixando o "como" para a própria rede. O usuário, nesse contexto, pode ser definido tanto como um operador/administrador da rede, como um usuário final. Esta camada de abstração deve facilitar o gerenciamento da rede, aumentando a eficiência operacional e reduzindo a dependência de gerenciadores humanos técnicos, aumentando a escalabilidade da provisão de NaaS.

Intent (intenção), no contexto de IBN, pode ser definido como um conjunto de objetivos operacionais (*i.e.,* que a rede deve implementar) e resultados (*i.e.,* que a rede deve entregar) definidos de forma declarativa sem explicitar como eles devem ser atingidos ou implementados (Clemm et al., 2022). Para explicitar de forma mais clara o quê, de fato, constitui uma intenção, são listados abaixo exemplos retirados do trabalho (Leivadeas e Falkner, 2023b):

- "Link utilisation in every link should be less than 70%."
- "My server application should be able to handle at least 10000 HTTP requests and a minimum bandwidth of 1 Gbps."
- "Route traffic of computer x through a Firewall."
- "Alice from the finance department requests all of her e-mails to any other department to be encrypted."

Observando os exemplos, fica claro que o nível de abstração de uma intenção é bastante variável. Diversos fatores contribuem para isso, sendo os principais o nível de conhecimento técnico do usuário e o contexto de negócio. Outra característica das intenções é que elas não

ficam estáticas em uma IBN, já que elas podem ser alteradas ou ainda removidas pelo próprio usuário.

Para que uma IBN seja completamente funcional, são necessários cinco componentes, com responsabilidades que vão desde o momento em que o usuário expressa sua intenção, até a implantação (*i.e., deployment*) e manutenção do serviço de rede. São eles: *Intent Profiling*, *Intent Translation*, *Intent Resolution*, *Intent Activation* e *Intent Assurance* (Leivadeas e Falkner, 2023b). A Figura 2.1 contém uma representação visual dos componentes da IBN e a comunicação entre cada um deles.

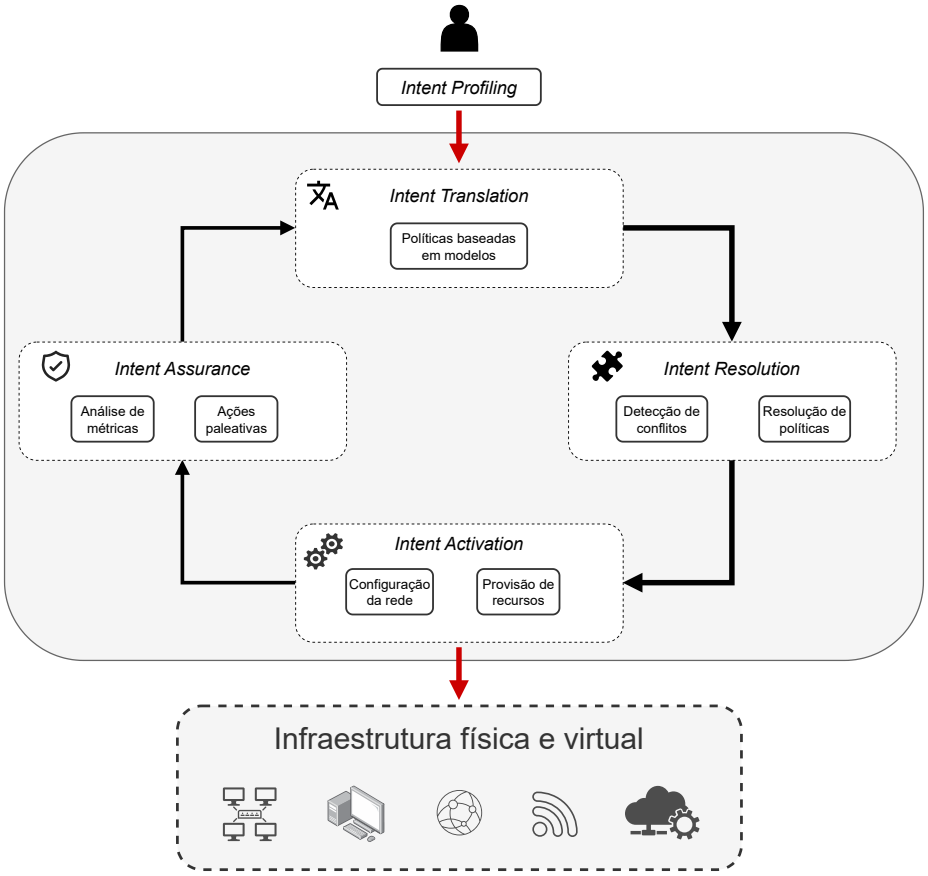


Figura 2.1: Componentes da IBN.

Cada componente é descrito em detalhes a seguir.

2.2.1 *Intent Profiling*

Intent Profiling ou perfilamento de intenções, define o início do ciclo de uma IBN, pois é nesta etapa em que o usuário define sua intenção para a rede, e onde efetivamente ocorre a interação com o usuário. Como já mencionado, a utilização de uma intenção para gerenciar a rede deve facilitar este processo. Portanto, a forma como o usuário interage com a rede não deve ser excessivamente técnica e complexa, mas simples e direta.

A forma de projetar este componente está condicionado a diversos aspectos que circundam o desenvolvimento da IBN, particularmente, o escopo da rede e seus usuários. Aqui, será apresentado resumidamente a classificação de intenções definida pela IEEE (*Institute of Electrical and Electronics Engineers*) no RFC 9316 (*Request For Comments*) (Li et al., 2022), e

as principais formas existentes de capturar a intenção do usuário no contexto de IBN (Leivadeas e Falkner, 2023b).

A taxonomia proposta pela IEEE define que a intenção pode ser classificada segundo: solução, tipo de usuário da intenção, tipo de intenção, escopo da intenção, escopo da rede, abstração e ciclo de vida. A solução da intenção é dividida em três, e indica os domínios de rede que a IBN deve suportar. Estes são: redes de operadoras, redes de *data center* e redes empresariais.

Os tipos de usuário estão fortemente relacionados com a solução da intenção, ou seja, com o domínio de rede. Pela natureza diversa dessa classificação, aqui serão apresentados apenas os principais tipos de usuário. Primeiramente, os clientes/usuários finais trabalham na mais alta camada de abstração, e geralmente não possuem conhecimento de detalhes técnicos – conhecimento que idealmente não deve ser relevante neste nível. Ainda assim, eles expressam políticas de alto nível relacionadas a suas necessidades de negócio.

Há também os desenvolvedores de aplicação, que trabalham em uma camada de abstração específica, o que significa que pode não haver um mapeamento claro entre sua forma de enxergar certos objetos e como eles são de fato configurados no contexto da rede. Por fim os operadores de rede são usuários com bastante conhecimento, tanto técnico quanto em relação à implementação da rede. No entanto, podem estar distantes da aplicação ou dos serviços fornecidos pelos usuários finais.

O tipo da intenção também pode ser considerado na categorização de intenção. Cada tipo descrito aqui possui um ou mais subtipos, e devem ser atendidos pela IBN: intenção de serviço ao consumidor, intenção de serviço de rede e de rede subjacente, intenção de rede e de rede subjacente, intenção de gerenciamento da nuvem, intenção de gerenciamento de recursos da nuvem, intenção de estratégia e intenção de tarefas operacionais.

O escopo da intenção aborda diversas categorias. Ele se refere à conectividade se o usuário cria ou modifica uma conexão. Envolve a segurança/privacidade se a intenção especifica alguma configuração de segurança para a rede ou para o usuário. Trata da aplicação se a intenção afeta alguma aplicação, e especifica as características QoS da rede.

Todas as intenções são aplicadas a um escopo específico da rede, que representa os elementos da infraestrutura afetados por aquela intenção. Por exemplo, considere uma intenção que determina o bloqueio de todo o tráfego SSH (*Secure Shell*) vindo da Internet para uma rede interna. Nesse caso, o *firewall* responsável pela filtragem entre esses domínios torna-se o escopo da intenção, pois é o componente da rede no qual a política será efetivamente aplicada.

O nível de abstração diz respeito tanto ao nível de abstração da intenção em si, quando ao nível de abstração necessário ao retornar um feedback para o usuário após a execução da intenção. Assim, o nível de abstração pode ser qualificado em técnico, quando informações de baixo nível e indicadores específicos de recursos da rede são mencionados, ou não-técnicos, quando o foco é meramente o sucesso ou não da aplicação da intenção e aspectos qualitativos mais simples.

O ciclo de vida pode ser persistente, ou seja, a partir do momento em que a intenção é ativada, ela é mantido na IBN até que seja removido/desativado pelo usuário, ou transiente, que significa que a intenção é expressada e ativada apenas uma vez, mas não é mantida depois disso.

Do ponto de vista da implementação, o componente de *Intent Profiling* pode ser desenvolvido de diversas formas, a depender do contexto específico. As formas mais relevantes são descritas a seguir.

Modelo/interface gráfica: dentre todas as formas de capturar a intenção do usuário, esta é a mais simples, pois limita as possibilidades de o usuário expressar sua intenção a um escopo pré-definido, o que também torna mais fácil as etapas subsequentes da IBN. De forma

básica, a interface gráfica permite que o usuário escolha opções já modeladas de intenção de acordo com diferentes categorias ou tipos.

Apesar de simples, esta opção ainda apresenta uma flexibilidade no que diz respeito a adaptar a interface para diferentes níveis de usuário. Por exemplo, a quantidade de informações presentes para um usuário de nível não-técnico pode ser reduzida, ao passo que um administrador de rede poderia ser apresentado um controle mais granular para expressar a intenção.

Pelo fato de o escopo ser delimitado por quem projeta a IBN, alguns desafios que outras formas de *profiling* enfrentam são atenuados. Por exemplo, delimitar com clareza os tipos de intenção que a rede suporta, uma vez que as intenções tendem a ser mais complexas quando se usa uma linguagem natural.

NLP (*Natural Language Processing*): outra forma de realizar o *intent profiling* é através do processamento de linguagem natural. Esta forma permite que o usuário se expresse em termos mais amplos e através de diferentes meios, como por comando de voz (Chaudhari et al., 2019) ou por uma interface no estilo *chatbot* (Jacobs et al., 2021).

Apesar de mais complexo, esse método apresenta maior dinâmica, possibilitando que haja um intercâmbio cíclico entre o usuário e a ferramenta para estabelecer a intenção. Em outras palavras, quaisquer erros contidos na intenção do usuário podem ser corrigidos pelo mecanismo de NLP, indicando ao usuário uma incoerência semântica, por exemplo. Outra vantagem é utilizar do próprio feedback do usuário como entrada para a melhoria do sistema.

2.2.2 *Intent Translation*

Intent Translation ou tradução de intenção: após o usuário expressar suas intenções, é necessário que estas sejam mapeadas efetivamente na rede. Portanto, nesta etapa, a intenção abstrata do usuário é traduzida em uma política de baixo nível que será adotada pelos dispositivos da rede.

A seguir, apresentam-se os os diversos métodos que podem ser utilizados para desenvolver este componente. Cada método é mais adequado para lidar com um escopo e tipo de intenção específica, portanto, a escolha entre cada um depende do contexto.

Template/Blueprint: a maneira mais simples de realizar a tradução é através de *templates* que, pré-definidos, são associados a uma intenção específica do usuário. Este método está associado à utilização de uma interface gráfica para o *profiling*.

Mapping: o mapeamento consiste na decomposição de uma intenção em classes e sub-classes de políticas de rede cada vez mais específicas, visando identificar o subdomínio mais adequado para atender aos requisitos da intenção.

Machine Learning: a versatilidade do aprendizado de máquina permite seu uso para traduzir intenções de diferentes formas. Pode-se utilizar, por exemplo, um modelo de *deep learning* treinado num banco de dados contendo intenções e políticas de rede associadas. Já o uso de NLP, por exemplo, pode permitir a análise sintática e semântica de termos usados em uma intenção, sendo apropriada para lidar com sinônimos e palavras fortemente correlacionadas.

2.2.3 *Intent Resolution*

Intent Resolution ou resolução de intenção: como uma IBN suporta múltiplas intenções, e até mesmo a ativação de novas intenções quando já existem intenções ativas, é necessário verificar se há alguma contradição interna entre a nova intenção e as já existentes. Não só isso, mas é necessário garantir que a intenção pode realmente ser implantada na rede. Estes dois aspectos, que compõe esse componente da IBN, são chamados, respectivamente, de detecção de conflitos (*Conflict Detection*) e resolução de políticas (*Policy Resolution*).

O primeiro passo para a detecção de conflitos é verificar a viabilidade da intenção. Esse passo inicial é necessário para evitar que intenções que não fazem sentido no contexto da rede passem adiante. Indicar, por exemplo, que a conexão entre dois *hosts* deve ter a largura de banda limitada não é possível caso um dos *hosts* não exista.

Após a primeira etapa de verificação, que leva em consideração principalmente a estrutura e a capacidade da rede, deve haver uma verificação de políticas. Esse passo consiste em analisar o estado atual da rede para detectar o conflito, e não sua infraestrutura e capacidade.

Ainda podem ser empregados mecanismos mais sofisticados de detecção de conflitos, que se baseiem, por exemplo, na utilização de grafos para representar as políticas da rede, ou ainda utilizando *Machine Learning*.

Após os conflitos serem detectados, surge a necessidade de resolvê-los, caso isso seja possível. Para alcançar isso, várias técnicas podem ser utilizadas. A mais simples delas, é por priorização (Leivadeas e Falkner, 2023b). Basicamente, cada intenção recebe um nível de prioridade, e, havendo um conflito entre duas intenções, a de prioridade maior prevalece. Esses níveis podem ser definidos explicitamente em cada intenção, ou através de regras de negócio implícitas, como priorizar políticas de grupo a individuais e intenções de operadores a de usuários finais.

Outras técnicas como o uso de lógica formal ou de enxergar a resolução do conflito como um problema de otimização podem ser utilizadas. Em última instância, no entanto, é ainda possível exigir que o usuário final ou o operador de rede se envolva para resolver o problema, tanto para decidir uma situação ou para agir em situações em que a IBNS (*Intent-Based Networking System*) não é capaz de fazer nada.

2.2.4 *Intent Activation*

Intent Activation ou ativação de intenção: tendo a garantia de que não haverá nenhum conflito interno ao adicionar uma intenção nova na rede, esta intenção deve ser provida de fato. Este componente realiza a implantação da intenção, reservando recursos computacionais necessários e automatizando o processo de configuração.

De maneira geral, o ambiente isolado das redes virtualizadas facilitam o processo de configuração efetiva da rede de acordo com as intenções do usuário (Cohen et al., 2013). No entanto, o processo de ativação deve refletir exatamente os requisitos, respeitando os limites gerais impostos pelo administrador da rede (*e.g.,* balanceamento de carga) e a topologia subsistente. Além disso, o processo de ativação de uma intenção não é estático, ou seja, devem ser considerados aspectos do estado atual da rede.

A literatura classifica a ativação em duas categorias principais. A primeira delas, é *Flow Activation*. Trata-se de intenções de conectividade, em especial em relação a selecionar o caminho entre dois *endpoints* e alocar os recursos de rede para esse caminho.

A outra categoria é denominada *SFC (Service Function Chaining) Activation*, e é mais focada em serviços complexos que exigem o uso de VNFs. Nesses casos, o trabalho da ativação está em prover os recursos computacionais exigidos para cada NFV.

Estas categorias de ativação situam-se em um contexto no qual a intenção deve ser ativada em cima de uma infraestrutura já existente. Não só isso, mas, muitas intenções mais simples, como definições de uma ACL (*Access Control List*), requerem apenas uma configuração direta de um dispositivo de rede (seja físico ou virtual), e casos como esse não se encaixam nas categorias citadas (Leivadeas e Falkner, 2023b).

2.2.5 *Intent Assurance*

Intent Assurance ou garantia de intenção: a rede não é estática, já que seu estado global está em constante mudança devido a variações de carga, movimentação de *hosts*, falhas de dispositivos e alterações dinâmicas em suas condições operacionais. Consequentemente, a rede deve ser continuamente monitorada para verificar se as intenções inicialmente expressas pelo usuário ainda estão sendo asseguradas. Caso alguma intenção não esteja mais atendida, a rede deve tomar as devidas medidas para solucionar este problema, seja de forma reativa ou ainda preemptiva. Esse problema é definido como *intent drift*, e ocorre quando a intenção que foi assegurada em algum momento deixou de ser, ou, ainda é assegurada, mas de forma menos efetiva (Clemm et al., 2022).

Esse componente é fortemente dependente de telemetria e análise de dados. Portanto, é essencial que exista um agente de monitoramento que faça o monitoramento constante da rede. Os dados monitorados devem estar relacionados às métricas pré-estabelecidas, que, por sua vez, devem ser pertinentes para garantir o *assurance* das intenções aplicadas na rede. A título de exemplo, a métrica de *delay* está fortemente relacionada à garantia de uma intenção como: "A latência entre os computadores A e B deve ser de no máximo X milissegundos".

Sendo assim, aqui serão descritas as principais ações que uma IBN pode executar para manutenção do *assurance* de acordo com duas categorias: reativo e proativo.

A principal ação tomada no contexto **reativo** é *flow migration* (Leivadeas e Falkner, 2023b). Essa técnica é utilizada quando a intenção lida com a conectividade entre pontos na rede. Sendo assim, ela está fortemente relacionada com as métricas de *delay*, *throughput* e/ou *link utilisation*. Na prática, o controlador SDN iria, assim que fosse detectado o *intent drift*, redirecionar o fluxo entre os nodos devidos da rede.

A migração de fluxo também pode ocorrer no caso de uma mudança de topologia. Esse caso, no entanto, é mais crítico, uma vez que intenções versam sobre nodos específicos da rede podem ser completamente inutilizadas, tornando-se necessário haver um processo de verificação adicional.

Outra forma de garantir o *assurance* é através da migração de VM (*Virtual Machine*) (Leivadeas e Falkner, 2023b). Essa técnica é adequada para lidar com intenções que dizem respeito a uma SFC, já que a QoS pode ser afetada intensamente pelos recursos computacionais alocados. Sendo assim, podem ser estabelecidos limiares referentes à utilização dos recursos, e, havendo uma ocorrência de quebra desses limiares, pode ser feita a migração da VM que hospeda as VNFs.

Outro método, alternativo à migração de VM, é o escalonamento de recursos (Leivadeas e Falkner, 2023b). Ele busca solucionar os mesmos problemas que a migração de VM resolve, porém, possui o benefício de evitar o *downtime* inerente à um processo de migração, aproveitando a facilidade de escalonar os recursos computacionais de uma rede virtualizada.

Em algumas situações, pode acontecer de a razão para uma intenção não estar sendo provida adequadamente ser uma falha geral na rede. Em casos como estes, é necessário usar o método de recuperação de falhas. Para atingir esse objetivo, podem ser empregados técnicas baseadas em grafos (Claise et al., 2023) ou redes neurais profundas (Yu et al., 2019; Yang et al., 2020).

Concomitantemente, pode ser empregado um esquema de priorização, já que em diversas situações é difícil encontrar soluções para todas as intenções afetadas em um tempo razoável.

Em casos onde nenhum mecanismo foi suficiente para assegurar a intenção, podem ser gerados relatórios que alertem o administrador da rede que certas intenções não estão sendo cumpridas. Estes relatórios podem ser gerados automaticamente de forma periódica de acordo com os requisitos da intenção.

O último mecanismo no contexto de *intent assurance* reativo é a reversão para um *Single Source of Truth* (SSoT). Um SSoT é o conjunto de intenções validadas na IBN (Clemm et al., 2022), e este conjunto, junto com os *records* dos estados operacionais da rede em um momento anterior às falhas ou *intent drifts* podem ser utilizados para reverter o estado da IBN. Essa tentativa de reversão pode ser considerada como uma ação de *self healing*, e, caso não seja bem sucedida, os *logs* desta operação são salvos e servem de referência para o administrador da rede.

Outro método de garantia de intenção que pode ser realizado, é a forma **proativa**. Neste método, ao invés de detectar desvios de intenção e tomar uma ação corretiva, técnicas, geralmente de *machine learning*, podem ser usadas para prever desvios futuros com base em dados históricos e realizar os devidos ajustes antes que a IBN seja impactada negativamente.

No contexto de *flow migration*, por exemplo, a tendência dos dados relacionados a uma métrica de QoS pode ser analisada com técnicas de *machine learning* e inteligência artificial (Saraiva et al., 2019) e indicar a necessidade de alguma ação a ser tomada, a migração de fluxo, nesse caso.

Outra ação que pode ser tomada no contexto de *assurance* proativo é o já mencionado escalonamento de recursos. Para alcançar isto, técnicas *machine learning* podem ser utilizadas para prever a utilização de recursos computacionais de uma VM no futuro, seja através de algoritmos de classificação ou por LSTM (*Long Short Term Memory*), e requisitar a alocação devida para evitar a superutilização quando esta for detectada.

Esse monitoramento ativo ainda pode ser utilizado, como um último recurso, para avisar os administradores da rede sobre problemas que provavelmente irão ocorrer no futuro.

3 TRABALHOS RELACIONADOS

Este capítulo busca apresentar soluções relacionadas à proposta deste trabalho, com um foco particular nas lacunas neles encontradas. Em particular são descritos trabalhos referentes à tradução das intenções do usuário no contexto da IBN, algumas propostas de arquiteturas específicas no contexto de redes IBN, e formas de definir topologias de rede.

3.1 TRADUÇÃO DE INTENÇÃO

Conforme aludido na introdução deste trabalho, existem diversas dificuldades para traduzir intenções de usuários expressas em linguagem natural para aspectos operacionais da rede – geralmente, utilizando de uma linguagem intermediária entre a intenção e a configuração em si. As principais dificuldades envolvem a falta de *datasets* que sustentem os modelos "clássicos" de aprendizado de máquina (*Machine Learning* - ML), o vasto vocabulário que pode ser utilizado para formar uma intenção e a individualidade de cada caso de uso (*e.g.,* as intenções em uma rede para um departamento de informática universitário difere grandemente de uma rede que implementa um sistema hospitalar).

Na tentativa de solucionar o problema da falta de dados de treinamento para se utilizarem as técnicas de ML, foi introduzido o artifício de incorporar o feedback do operador durante a expressão das intenções. Isso funcionaria a partir de uma interface estilo *chatbot*, na qual seria solicitado, a cada expressão de intenção, o parecer do operador se a intenção foi traduzida corretamente ou não. Ao longo do tempo, estes dados são coletados para aumentar o *dataset* inicial, permitindo que o sistema aprenda a traduzir intenções com maior corretude.

Este mecanismo foi implementado em Jacobs et al. (2021), no qual os autores desenvolveram o sistema Lumi. Esse sistema, no entanto, ainda padece de alguns problemas. Primeiramente, o operador do sistema deve ter uma familiaridade com uma linguagem intermediária de configuração de rede criada para o Lumi, já que o mecanismo de feedback se baseia em mostrar ao operador como a intenção foi interpretada pelo sistema nesta linguagem. Não só isso, mas a dependência na interação com o operador pode atrasar a definição das intenções, além de haver possivelmente uma curva de aprendizado para o sistema alcançar o nível de acurácia desejável.

3.2 PROPOSTAS DE ARQUITETURA

Além das dificuldades citadas na seção anterior, a propagação da IBN ainda é desacelerada pela falta de uma arquitetura genérica que guie uma implementação, e pela própria falta de uma implementação genérica em si. Os esforços acadêmicos despendidos no desenvolvimento de uma arquitetura foram colocados em arquiteturas que podem ser classificados em duas categorias: fortemente acopladas a *frameworks* já existentes, ou voltadas para casos de uso demasiado específicos.

O principal exemplo de uma arquitetura acoplada, é a proposta em Yu et al. (2024), na qual os autores apresentam uma arquitetura que, embora descrita como abrangente, é especificamente implementada sobre o ONAP (*Open Network Automation Platform*), um *framework open-source* para a orquestração e a automação de redes (ONAP, 2025).

A utilização do ONAP faz com que a arquitetura seja composta por uma integração de diversos microsserviços desta plataforma, o que gera uma forte dependência a eles. Isto

implica que quaisquer limitações que estejam presentes no *core* do ONAP acabariam por impactar diretamente a arquitetura e a implementação (Yu et al., 2024). Além disso, a complexidade inerente do ONAP exige do operador da rede um conhecimento técnico específico da arquitetura, gerando um obstáculo adicional para a adesão a este sistema.

Outra barreira presente na proposta é a falta de clareza sobre como a tradução das intenções é realizada. Nesse sentido são propostos dois módulos baseados em NLP que podem ser carregados alternadamente. Um deles utiliza a técnica *Question Answering* (QA), comparando os modelos CoT-T5 e Flan-T5, e o outro NER (*Named Entity Recognition*), comparando os modelos BERT-CRF (*Bidirectional Encoder Representations from Transformers* e *Conditional Random Field*) e BERT-Span. O problema, no entanto, reside no fato de não haver menção alguma sobre os dados de treinamento utilizados para nenhum desses modelos. Isso torna impossível a validação dos resultados e acaba evidenciando a natureza "caixa-preta" desta solução, já que organizações que queiram adotá-la não seriam capazes de replicar o treinamento dos modelos, e possivelmente não seria possível adotar seus próprios dados para isso.

Já em Njah et al. (2025) é descrita uma arquitetura para IBN orientada por IA. A arquitetura é dividida em três componentes. O primeiro componente é o *Intent Refinement*, que consiste na tradução de intenções definidas em linguagem natural para modelos de políticas de rede, através do uso de LLMs, que são refinadas para serem posteriormente mapeadas em *scripts* de configuração de rede de baixo nível. Em seguida, o segundo componente é o *Intent-Policy Activation*, que envolve primariamente a resolução dos conflitos que surgem entre as intenções e o estado da rede. Para isso, propõe-se a utilização de técnicas de *Deep Reinforcement Learning* (DRL). Por fim, o componente *Intent-Policy Assurance* busca garantir as intenções através de agentes-DRL baseados no *framework* MADRL (*Multi-Agent Deep Reinforcement Learning*). Estes agentes interagem com a infraestrutura da rede, adotando, de forma colaborativa, estratégias para a garantia das diversas intenções ativas.

Apesar da robustez desta arquitetura, ela se limita a uma proposta primariamente conceitual, já que os autores não apresentam uma implementação. Além disso, um dos principais componentes, o *Intent-Policy Assurance*, é admitidamente um "desenvolvimento em andamento" e sua análise de desempenho, ponto fundamental para validar a proposta, é explicitamente deixada para um trabalho futuro. Tudo isso torna difícil enxergar a viabilidade de uma implementação baseada nesta proposta.

3.3 DEFINIÇÃO DE TOPOLOGIAS

O JMP (Santana et al., 2017) é uma solução capaz de interpretar arquivos JSON com definições de topologias de rede, traduzí-las e executá-las em código Mininet. O JMP foi criado com objetivo de adicionar uma camada de abstração entre o usuário e a plataforma de emulação de rede (Mininet). A Tabela 3.3 apresenta as explicações e descrições de cada um dos possíveis campos do JSON.

O JMP pode ser utilizado como um ponto de partida para o desenvolvimento de uma IBN, já que suas funcionalidades podem auxiliar no processo de implantação através da interpretação do JSON em código Mininet, abstraindo essa etapa. Além disso, o JSON de entrada pode servir como um SSoT para o sistema, assim como foi utilizado neste trabalho.

3.4 CONCLUSÃO

Como evidenciado neste capítulo, os trabalhos existentes na literatura promovem soluções que avançam o campo de estudo da IBN, mas ainda apresentam certas lacunas. Estas

Tabela 3.1: Campos do JSON de Topologia do JMP

Campo			Tipo	Obrigatório	Descrição
ID			String	Sim	Identificador da topologia.
COMPONENTS	HOSTS	ID	String	Sim	Nome único do host.
		IP	String	Sim	Endereço IP do host.
	SWITCHES		Array (Strings)	Não	Lista de IDs de switches.
	CONTROLLERS		Array (Strings)	Não	Lista de IDs de controllers.
	OVSSWITCHES	ID	String	Sim	Identificador do switch OVS.
		CONTROLLER	String	Sim	ID do controller associado.
CONNECTIONS	IN/OUT		String	Sim	ID do primeiro endpoint.
	OUT/IN		String	Sim	ID do segundo endpoint.

lacunas são um impedimento para uma difusão mais ampla da IBN como paradigma de redes nos ambientes empresariais e acadêmicos.

Em particular, são mais notórios problemas como a existência de arquiteturas pouco genéricas, focadas apenas em um componente, a falta de uma implementação genérica e extensível, arquiteturas acopladas fortemente a outros sistemas, ou soluções que não resolvem suficientemente os desafios encontrados para a tradução das intenções.

Além disso, encontram-se soluções que não são diretamente voltadas para a área da IBN, no entanto, se desenvolvidas, podem servir de base para ela.

Diante deste cenário, o próximo capítulo detalha a proposta desenvolvida com o propósito de procurar a solução de algumas das lacunas mostradas neste capítulo, em particular, a proposta de uma arquitetura genérica;

4 UMA ARQUITETURA ORIENTADA A *PLUGINS* PARA REDES BASEADAS EM INTENÇÃO

Desde o início do estudo do campo de IBN, pouco esforço foi colocado na tentativa de criar um sistema IBN completo ou um *framework* que permita a criação de tal sistema. Um dos principais desafios que impedem sua ampla adoção é a falta de padronização ou de uma arquitetura genérica que possa ser utilizada em diferentes contextos. Sem um padrão que especifique as operações necessárias que uma IBN deve contemplar, a criação de tais sistemas torna-se completamente individualizada, onde cada empresa implementa a sua própria solução com base em seus componentes e casos de uso específicos.

Nesse sentido, este trabalho propõe a PIÁ (*Plugin-based Intent Architecture*), uma arquitetura genérica baseada em *plugins* para IBN. Essa arquitetura é descrita na Seção 4.1, junto de uma explicação de cada um de seus componentes. Já a Seção 4.2 descreve o protótipo implementado com base nesta arquitetura.

4.1 ARQUITETURA

Uma arquitetura completa para IBN deve abranger desde a definição da intenção do usuário, até a sua implantação e garantia contínua na rede. A Figura 4.1 mostra a arquitetura proposta, que abrange todos esses aspectos.

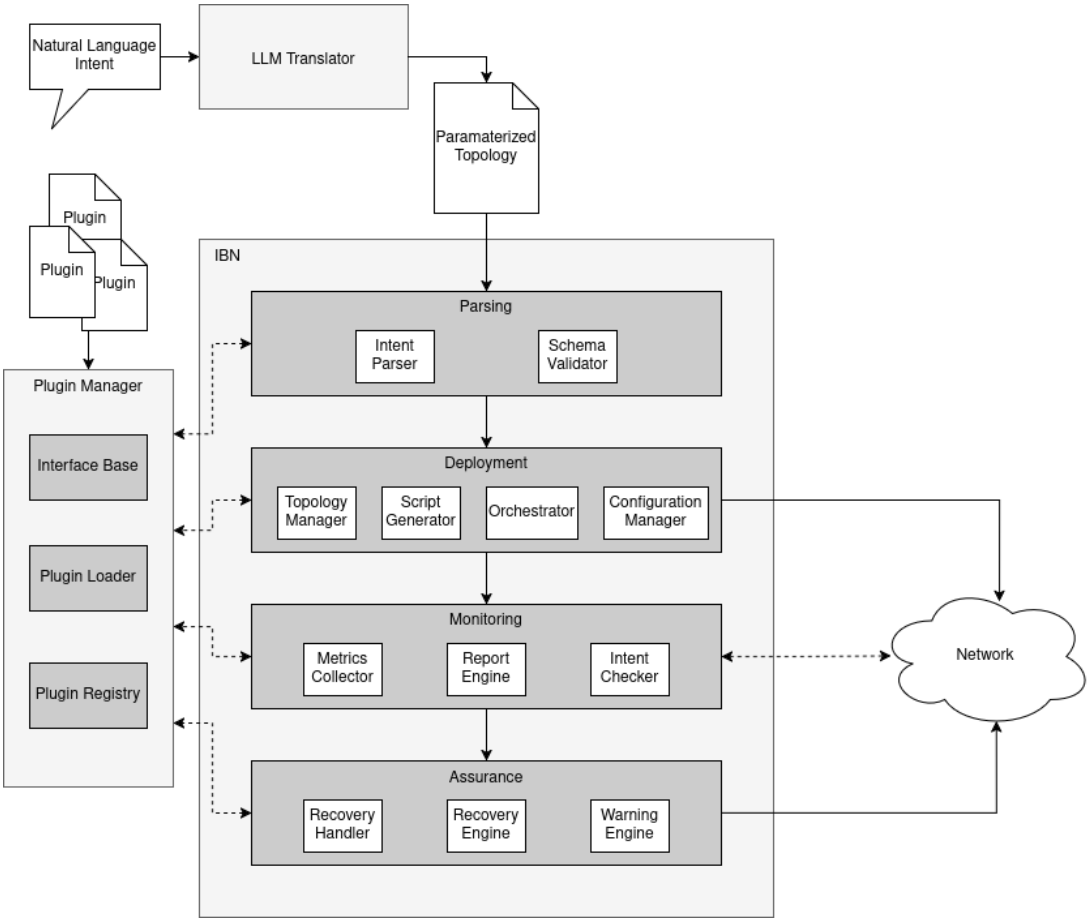


Figura 4.1: Visão geral da arquitetura PIÁ.

O ponto de início é a intenção do usuário, que pode ser definida em linguagem natural e passada por um tradutor LLM. As intenções específicas que o usuário pode definir são particulares de cada implementação. A intenção do usuário pode também ser definida diretamente no arquivo descritivo da topologia de rede, denominado nessa arquitetura de PPT (*Piá's Parameterized Topology*), que serve como entrada e uma "fonte de verdade única" (SSoT) para o sistema IBN. PPT pode ser definido de diversas formas, desde que seja estruturado de forma a habilitar a expressão de todas as possíveis intenções do usuário. A estrutura da PPT pode ser baseada em formatos já existentes, como JSON e YAML (YAML *Ain't Markup Language*), como também pode ser definido um formato específico para a implementação. A PPT é central para a arquitetura, já que ele representa uma SSoT, que será a base para todas as outras operações dentro do sistema IBN em si.

A definição da PPT, que descreve a topologia de rede, pode ser realizada diretamente por um usuário que possua o conhecimento técnico e seja capaz de utilizar a linguagem estruturada, ou por um tradutor LLM, que, por sua vez, irá traduzir uma intenção definida em linguagem natural para este arquivo estruturado.

O primeiro componente da arquitetura PIÁ é o ***Parsing***, responsável por estabelecer a fronteira entre o resto do sistema e a PPT definido pelo usuário. Como o próprio nome implica, este componente é responsável por interpretar as intenções do usuário e validá-las. Estas duas responsabilidades são atribuídas, respectivamente, aos subcomponentes *Intent Parser* e *Schema Validator*. Ambos atuam juntos no processamento das intenções, sendo que o *Intent Parser* deve interpretar cada componente seguindo a estrutura definida no próprio arquivo, e armazená-lo dentro das estruturas de dados adequadas dentro do sistema para que estas possam ser utilizadas adequadamente nas etapas subsequentes. Por sua vez, o *Schema Validator* deve realizar a detecção de erros ao analisar as intenções, verificando inicialmente se há consistência léxica e semântica. Caso sejam detectados erros, estes devem ser acusados ao operador, e, dependendo da gravidade, abortar a execução indicando a impossibilidade de levá-la adiante.

Após o processamento das intenções dentro do sistema pelo componente de *Parsing*, deve atuar o componente de ***Deployment***. O *Topology Manager* é o subcomponente encarregado de gerenciar a topologia que foi gerada, criando internamente: (*i*) abstrações para cada elemento da rede (*e.g.,* um *switch* pode ser armazenado como uma variável de um tipo customizado); (*ii*) as conexões entre os dispositivos e; (*iii*) parâmetros sobre estas conexões, conforme o que foi processado no *Parsing*. O *Script Generator* deve, a partir da topologia, gerar um *script* ou artefato da implantação. O artefato por sua vez deve ser executado, ação que é realizada pelo *Orchestrator*. Já o *Configuration Manager* é o subcomponente responsável por aplicar as configurações de dispositivo e conexão especificadas (*e.g.,* IPs dos *hosts*) na PPT.

Feita a implantação da rede, inicia-se o processo de monitoramento, realizado pelo componente ***Monitoring***. A parte mais fundamental deste processo é coletar as métricas que estão relacionadas a cada uma das intenções citadas, para que se verifique se estas estão sendo atendidas. Um exemplo seria uma intenção que limite a utilização de um enlace de comunicação. Nesse caso, é necessário coletar periodicamente os dados desse enlace. A coleta das métricas pode ser realizada por um sistema centralizado que possui o controle direto sobre os dispositivos da rede e os monitora ativamente, ou de modo descentralizado, onde cada um dos dispositivos da rede será encarregado de coletar localmente e enviar estes dados a um sistema central que irá processá-los. A periodicidade da coleta das métricas pode ser definida individualmente, segundo a especificidade da métrica que está sendo avaliada.

Outra peça de importância fundamental no *Monitoring* é o subcomponente *Intent Checker*, responsável por verificar se a intenção está ou não sendo assegurada. Este componente está fortemente relacionado com o *Metrics Collector*, já que receberá dele seus dados de entrada.

Recebidos estes dados, eles serão comparados com as intenções definidas inicialmente pelo usuário. Seguindo o exemplo mencionado anteriormente, caso o usuário especifique que a utilização de um enlace deve ser limitada a 70% da capacidade total, os números reais de utilização serão comparados com o limite imposto. Em alguns casos, pode ser que a intenção seja mais abstrata, e é justamente nos componentes da tradução e do *Parsing* que elas devem ser processadas em uma informação operacional e quantificável do ponto de vista da rede.

O último subcomponente do *Monitoring* é o *Report Engine*, que deverá coordenar a coleta das métricas e a verificação das intenções para gerar *logs* e relatórios de execução. A natureza desses relatórios pode ser múltipla, dependendo do caso de uso requerido (*e.g.,* há a possibilidade de ser construído apenas com base nas métricas), mas sempre deve ser disponibilizado para o usuário. Pode-se considerar ainda, em certos casos, que a própria forma de monitorar a rede pode ser uma intenção (Clemm et al., 2022).

O último componente do sistema principal da IBN é o ***Assurance***, que atua em conjunto com o *Monitoring* para a garantia das intenções. O primeiro subcomponente é o *Recovery Handler*, que atua como um orquestrador diante de uma intenção que foi violada. É nesse subcomponente que se avalia inicialmente se é possível recuperar uma intenção violada. Sendo esse o caso, ele deve considerar, dentre as possíveis formas de recuperação, qual é a mais adequada, levando em consideração o estado atual da rede, a topologia e as demais intenções estabelecidas pelo usuário. Feita essa escolha, a *Recovery Engine* é acionada com o intuito de ativar os mecanismos de recuperação. Em casos de impossibilidade de recuperação, deve ser acionada a *Warning Engine*, que emite um alerta ao operador da rede informando sobre a quebra da intenção. Nestes casos, é necessário uma intervenção direta do operador da rede para solucionar o problema reportado.

Dentro da *Recovery Engine* são implementados os diversos mecanismos de recuperação da rede, que podem envolver ações como a reaplicação de configurações de rede ou re-roteamento de tráfego. Estas ações podem ser implementadas como *scripts* nos quais são definidos todos os passos de recuperação. O importante é que este subcomponente seja projetado para ser capaz de recuperar todas as intenções definidas pelo usuário. No entanto, em redes dinâmicas, é altamente improvável que as intenções sejam recuperadas em sua totalidade. Nestes casos, atua a *Warning Engine*, estabelecendo principalmente a conexão entre a rede e o usuário. A forma como o usuário será engajado, no entanto, é variável – pode ser exigida dele a priorização de uma intenção sobre outra, ou ele pode apenas ser alertado da impossibilidade de recuperação. Em todo o caso, esses dois subcomponentes são coordenados pelo *Recovery Handler* de modo a lidar com quaisquer violações de intenção.

Além de todos estes componentes internos que compõem o sistema IBN principal, há um sistema externo chamado *Plugin Manager*. Este componente, como o nome sugere, é um gerenciador de *plugins*, ou seja, de extensões que podem ser acopladas ao sistema principal, adicionando funcionalidades que não são implementadas pelo sistema base. Com a adição de um *plugin*, existem parâmetros adicionais que podem ser especificados em um local específico da PPT e que também serão processado pelo componente *Parsing*. Estes parâmetros extras serão discutidos em mais detalhes na Seção 4.2. A necessidade de haver um módulo desse tipo encontra-se na tentativa de manter o sistema base enxuto e genérico e, ao mesmo tempo, permitir que o sistema seja capaz de atender a intenções relacionadas a contextos e casos de uso mais específicos. Por exemplo, considere que o usuário deseja expressar uma intenção relacionada à instanciação de um servidor *web* replicado. Nesse contexto específico onde os parâmetros da PPT não são suficientes para expressar a intenção do usuário, é necessário estender a arquitetura base através de *plugins* que implementem as funcionalidades correspondentes.

O primeiro componente do sistema *Plugin Manager* é denominado de Interface Base. Essa interface expõe as funções que todo *plugin* deverá implementar, como no contexto de programação orientada a objetos. A Interface Base garante que a funcionalidade padrão do sistema seja estendida através da implementação de *plugins*, evitando a necessidade de modificar aspectos internos. Pelo fato de os *plugins* estenderem a funcionalidade base, eles estão fortemente ligados com os componentes internos principais da IBN (*i.e., Parsing*, *Deployment*, *Monitoring* e *Assurance*). Se fosse criado um *plugin* que adicionasse a funcionalidade de *firewall*, seria primeiro necessário definir os parâmetros deste *plugin*, que estão ligados ao *Parsing*. Depois, estabelecer a forma de monitoramento destes parâmetros, que se relaciona ao *Monitoring*. Por fim, definir ainda como garantir estes parâmetros, indicando funções de recuperação, que estão ligadas ao *Assurance*. Então, a *Interface Base* deve, com as funções, permitir que os *plugins* implementem cada um destes aspectos.

O segundo componente é o *Plugin Loader*, que conceitualmente é bastante simples – ele simplesmente carrega os *plugins* definidos nos arquivos externos e realiza a instanciação dentro do sistema principal. Por fim, o componente *Plugin Registry*, deverá armazenar internamente cada um dos *plugins* que foram carregados pelo *Plugin Loader*.

4.2 IMPLEMENTAÇÃO

Essa seção descreve como foi feita a implementação da arquitetura PIÁ[1]. O código foi desenvolvido em Python, implementando todos os componentes da arquitetura descrita na subseção anterior. Para a implantação e gerenciamento das redes, foi utilizado a Mininet, em particular sua API para Python. Há ainda um módulo adicional que utiliza a API do Gemini (Team et al., 2023) para a tradução de intenções expressas em linguagem natural.

Na Subseção 4.2.1, é descrita a implementação do `translator.py`, que faz a tradução da intenção do usuário em linguagem natural para JSON, bem como a estrutura da PPT. Já na Subseção 4.2.2, é descrita a implementação do programa principal `parser.py`, que faz o *parsing* e o *deployment*. Por último, na Subseção 4.2.3, é descrita a implementação do `monitor.py`, que faz o monitoramento e o *assurance*.

4.2.1 Tradução

A PPT é a base para o funcionamento do sistema. Tanto a PPT quanto o *parser* foram inspirados no JMP (Santana et al., 2017). O JMP é uma ferramenta de *parsing* e *deploy* de redes utilizando a ferramenta Mininet. Os campos existentes no JMP, no entanto, são bastante limitados, sendo somente possível especificar `ID`, `IP` e `CONTROLLER`. Por isso, foi adicionado o campo `PARAMS` em todos os componentes que existiam na JMP, tornando a rede mais customizável. Além disso, foi adicionado o campo `PLUGINS`, que permite a adição de qualquer novo parâmetro que o usuário implementar. A PPT descreve todos os componentes, conexões e *plugins* a serem utilizados. A PPT foi definido em formato JSON e está estruturado da seguinte maneira:

Código 4.1: "Estrutura da PPT (*Piá's Parameterized Topology*)."

```
{
   "ID": "Exemplo",
   "VERSION": "1.0",
   "DESCRIPTION": "Uma topologia de exemplo",
   "COMPONENTS": {
      "HOSTS": [
```

[1]Código disponível em `https://github.com/caiohrr/tcc-ibn`

```
            {
                "ID": "h1",
                "IP": "192.168.1.1/24",
                "MAC": "00:00:00:00:00:11"
            },
            {
                "ID": "h2",
                "IP": "192.168.1.2/24",
                "MAC": "00:00:00:00:00:12"
            }
        ],
        "SWITCHES": [
            {
                "ID": "s1",
                "TYPE": "OVSSwitch",
                "PARAMS": {
                    "PROTOCOLS": "OpenFlow13"
                }
            }
        ],
        "CONTROLLERS": [
            {
                "ID": "c0",
                "TYPE": "Controller",
                "PARAMS": {
                    "IP": "127.0.0.1",
                    "PORT": 6653
                }
            }
        ]
    },
    "CONNECTIONS": [
        {
            "ENDPOINTS": ["h1", "s1"],
            "PARAMS": { "BANDWIDTH": 200, "DELAY": "2ms" }
        },
        {
            "ENDPOINTS": ["h2", "s1"],
            "PARAMS": { "BANDWIDTH": 100, "DELAY": "3ms", "LOSS": 0.5 }
        }
    ]
}
```

A Tabela 4.2.1 descreve cada campo possível da PPT. A seguir, os principais campos são explicados em detalhes.

Components: O subcampo `HOSTS` pode conter as chaves `HOSTS`, `SWITCHES`, `CONTROLLERS` e componentes personalizados adicionados por *plugins*. No campo dos `HOSTS`, é necessário identificar o *host* com um ID, mas os subcampos de `IP` e `MAC` não são obrigatórios.

Código 4.2: Exemplo da estrutura `HOSTS` no JSON.

```
"HOSTS": [ { "ID": "h1", "IP": "10.0.0.1/24", "MAC": "00:00:00:00:00:01" }
    ]}
```

Igualmente ao campo anterior, os campos `SWITCHES` e `CONTROLLERS` possuem somente o subcampo `ID` obrigatório. Ambos possuem os subcampos `TYPE` e `PARAMS`, que identificam o tipo do componente e os parâmetros extras, respectivamente.

Campo			Tipo	Obrigatório	Descrição
ID			String	Não	Identificador da topologia.
VERSION			String	Não	Versão da PPT.
DESCRIPTION			String	Não	Descrição da topologia.
COMPONENTS	HOSTS	ID	String	Sim	Nome único do host.
		IP	String	Não	Endereço IP.
		MAC	String	Não	Endereço MAC
	SWITCHES	ID	String	Sim	Identificador do switch.
		TYPE	String	Não	Tipo (OVSKernelSwitch ou UserSwitch).
		PARAMS	Objeto	Não	Parâmetros extras.
	CONTROLLERS	ID	String	Sim	Identificador do controlador.
		TYPE	String	Não	Tipo (Controller ou RemoteCotroller)
		PARAMS	Objeto	Não	Parâmetros extras (IP, PORT, etc.).
CONNECTIONS	ENDPOINTS		Array (2)	Sim	IDs dos elementos conectados.
	PARAMS		Objeto	Não	Configurações do link (ex: BANDWIDTH, DELAY, LOSS).
PLUGINS			Array	Não	Lista de plugins aplicados à topologia.

Código 4.3: Exemplo das estruturas `SWITCHES` e `CONTROLLERS` no JSON.

```
"SWITCHES": [ { "ID": "s1", "TYPE": "OVSKernelSwitch", "PARAMS": { "
    PROTOCOLS": "OpenFlow13" } } ]

"CONTROLLERS": [ { "ID": "c0", "TYPE": "RemoteController", "PARAMS": { "IP":
     "127.0.0.1", "PORT": 6653 } } ]
```

Componentes personalizados: Para a adição de um parâmetro personalizado, é necessário um *plugin* registrado que implemente `ComponentPlugin` e declare o nome do componente (*e.g.,* `FIREWALLS`).

Código 4.4: Exemplo de um componente personalizado (`FIREWALLS`).

```
"FIREWALLS": [ { "ID": "fw1", "RULES": ["allow tcp", "deny udp"] } ]
```

Connections: Diferentemente dos outros campos, `CONNECTIONS` possui somente dois subcampos: `ENDPOINTS` e `PARAMS`. `ENDPOINTS` é obrigatório e do tipo *array*, que aceita somente dois elementos: o `ID` de cada um dos *hosts* da conexão que está sendo definida. Já o subcampo `PARAMS` não é obrigatório. Ele define as configurações do *link* (*e.g.,* `BANDWIDTH`, `DELAY`, `LOSS`).

Código 4.5: Exemplo da estrutura `CONNECTIONS` no JSON.

```
"CONNECTIONS": [ { "ENDPOINTS": ["h1", "s1"], "PARAMS": { "BANDWIDTH": 10,
    "DELAY": "5ms" } } ]
```

Tradução: Para auxiliar o usuário na criação das PPTs, foi criado o `translator.py`. Esse programa utiliza a API do LLM Gemini para fazer a tradução das intenções de linguagem natural para o formato JSON válido. A intenção do usuário é lida e é feita uma requisição à API juntamente com uma instrução do sistema, definida anteriormente no `translator.py`. Uma PPT é retornado após a requisição feita à API.

4.2.2 *Parser*

O programa `parser.py` é responsável por fazer o *parsing* da PPT e gerar um *script* Python que faz o *deploy* da rede na Mininet.

Ao ser iniciado, `parser.py` instancia a `PluginInterface`, que é a interface base para todos os tipos de *plugin*. Em seguida, são definidas as classes de *plugins* `TopologyPlugin`, `ScriptGeneratorPlugin`,`ComponentPlugin` e `MonitorRecoveryPlugin`. O Código 4.6 mostra a implementação dessas classes.

Após essas definições, a classe `PluginManager` é ativada. Ela importa e carrega todos os *plugins* que estão no diretório `plugins/`.

Código 4.6: Interfaces base dos *plugins*.

```
# ...
class PluginInterface(ABC):
    """Base interface for all plugins."""
    @abstractmethod
    def get_name(self) -> str: pass
    # ...

class TopologyPlugin(PluginInterface):
    """Base class for topology manipulation plugins."""
    @abstractmethod
    def process_topology(self, topology: 'Topology', params: Dict[str, Any])
        -> None: pass
```

```
12
13 class ScriptGeneratorPlugin(PluginInterface):
14     """Base class for script generation plugins."""
15     @abstractmethod
16     def generate_imports(self) -> List[str]: pass
17
18     @abstractmethod
19     def generate_post_start_code(self, topology: 'Topology', params: Dict[
           str, Any]) -> List[str]: pass
20     # ...
21
22 class ComponentPlugin(PluginInterface):
23     """Base class for custom network component plugins."""
24     @abstractmethod
25     def parse_component(self, component_data: Dict[str, Any]) -> Dict[str,
           Any]: pass
26
27     @abstractmethod
28     def generate_component_code(self, component: Dict[str, Any]) -> List[str
           ]: pass
29
30 class MonitorRecoveryPlugin(PluginInterface):
31     """Base class for intent monitor and recovery plugins."""
32     @abstractmethod
33     def get_check_functions(self) -> Dict[str, Callable]: pass
34
35     @abstractmethod
36     def get_recovery_functions(self) -> Dict[str, Callable]: pass
```

Depois da importação e carregamento dos *plugins*, o programa interpreta a topologia solicitada pelo usuário. Em seguida, a classe `Topology` é instanciada e inicia o processamento dos dados, fazendo a tradução dos componentes e suas configurações. Além disso, ele também executa os *plugins* e realiza o *parse* dos componentes customizados. O Código 4.7 mostra a implementação da função que interpreta estes componentes.

Código 4.7: Método `_parse_custom_components` da classe Topology.

```
1  # ...
2      def _parse_custom_components(self, components: Dict[str, Any]) -> Dict[
           str, List[Dict[str, Any]]]:
3          """Parse custom components using registered plugins."""
4          custom_components = {}
5
6          for component_type, component_data in components.items():
7              if component_type not in ["HOSTS", "SWITCHES", "CONTROLLERS"]:
8                  if component_type in self.plugin_manager.component_plugins:
9                      plugin = self.plugin_manager.component_plugins[
                           component_type]
10                     custom_components[component_type] = []
11
12                     if isinstance(component_data, list):
13                         for item in component_data:
14                             parsed = plugin.parse_component(item)
15                             custom_components[component_type].append(parsed)
16                     else:
17                         parsed = plugin.parse_component(component_data)
18                         custom_components[component_type].append(parsed)
19
```

```
20          return custom_components
21  # ...
```

Após o objeto de topologia ser totalmente processado, ele é entregue ao `MininetScriptGenerator`. Esta classe gera o *script* Python final, inserindo o código dos *plugins* nos lugares apropriados. O método `generate` coordena essa construção, como podemos ver no Código 4.8, abaixo.

Código 4.8: Método `generate` do `MininetScriptGenerator`.

```
1   # ...
2   def generate(self, topology: Topology, output_file: str = "topology.py"):
3
4       # Get plugin additions
5       plugin_additions = self.plugin_manager.get_script_generator_additions(
6           topology, topology.plugins_config
7       )
8
9       with open(output_file, "w+", encoding='utf-8') as mn_file:
10          # Write header and imports
11          self._write_header(mn_file, topology)
12          self._write_imports(mn_file, plugin_additions["imports"], ...)
13
14          # ... (network initialization) ...
15
16          # Pre-network plugin code
17          for line in plugin_additions["pre_network"]:
18              mn_file.write(f"\t{line}\n")
19
20          # ... (add standard components) ...
21
22          # Add custom components via plugins
23          self._write_custom_components(mn_file, topology)
24
25          # Start network
26          mn_file.write("\tnet.start()\n\n")
27
28          # Post-start plugin code
29          for line in plugin_additions["post_start"]:
30              mn_file.write(f"\t{line}\n")
31
32          # ...
```

4.2.3 *Monitoring* e *Assurance*

O programa `monitor.py` é o responsável por fazer o monitoramento e o *assurance* da rede. Quando em execução, a classe `IntentMonitor` carrega os *plugins* e inicia um laço de repetição, verificando se todas as intenções estão sendo atendidas a cada iteração. Para isso, no método `_monitor_loop`, são utilizadas funções específicas para cada tipo de intenção. O Código 4.9 mostra a implementação desse método.

Código 4.9: Método `_monitor_loop` da classe `IntentMonitor`.

```
1   # ...
2   def _monitor_loop(self):
3       """The main loop that checks all intents."""
4       if not self._monitoring_active:
```

```
5           return
6
7       # ...
8
9       for intent in self.intents:
10          check_function = self.check_functions.get(intent['type'])
11          # ...
12          try:
13              is_ok = check_function(intent)
14
15              if not is_ok:
16                  intent['status'] = 'BROKEN'
17                  # ... (logging) ...
18
19                  if self.recovery_enabled:
20                      recovery_function = self.recovery_functions.get(intent['type'
                            ])
21                      if recovery_function:
22                          print(f" -> Attempting recovery for '{intent['type']}'...
                                ")
23                          recovery_function(intent)
24              # ...
25      # Schedule the next check
26      self._timer = threading.Timer(self.monitor_interval, self._monitor_loop)
27      self._timer.start()
28  # ...
```

O trecho de código a seguir mostra como é feito o monitoramento da conectividade entre dois *hosts*. Nesse exemplo, a conectividade é checada através do comando `ping`. Se o comando retorna `0% packet loss`, a conectividade é considerada válida. Caso contrário, retorna `False` e em seguida o laço de repetição identifica a quebra na intenção, iniciando então a tentativa de recuperação.

Código 4.10: Função `check_connectivity`.

```
1  def check_connectivity(self, intent):
2      """Checks if two hosts can ping each other."""
3      host1_id, host2_id = intent['target']
4      host1 = self.net.get(host1_id)
5      host2 = self.net.get(host2_id)
6      result = host1.cmd(f'ping -c 1 {host2.IP()}')
7      is_successful = '0% packet loss' in result
8
9      return is_successful
```

Neste exemplo, a função de recuperação tenta habilitar as interfaces de rede dos *hosts*, usando o comando `ip link set {interface} up`, garantindo que as interfaces não estejam desativadas. Se a operação não for bem sucedida, a função retornará um erro e o monitor marcará a intenção como `BROKEN` e tentará executar a função de recuperação novamente na próxima iteração do loop.

Código 4.11: Função `recover_connectivity`.

```
1  def recover_connectivity(self, intent):
2      """Attempts to recover connectivity by ensuring host interfaces are 'UP'.
           """
3      host1_id, host2_id = intent['target']
4      host1 = self.net.get(host1_id)
```

```
5       host2 = self.net.get(host2_id)
6       try:
7           iface1 = host1.intfNames()[0]
8           iface2 = host2.intfNames()[0]
9           print(f" -> ACTION: Ensuring interfaces are UP for {host1_id}({iface1
                }) and {host2_id}({iface2}).")
10          host1.cmd(f"ip link set {iface1} up")
11          host2.cmd(f"ip link set {iface2} up")
12      except Exception as e:
13          print(f" -> ERROR: Failed to bring interfaces up for {host1_id}{
                host2_id}: {e}")
```

5 EXPERIMENTOS

Em vista de avaliar o protótipo implementado para a arquitetura PIÁ, foram realizados dois experimentos quantitativos. De maneira geral, o primeiro experimento tem como objetivo medir a acurácia do módulo de tradução de intenções, e o segundo, a capacidade de detecção e recuperação de quebras de intenção.

Ambos os experimentos foram executados em uma máquina com a distribuição Arch Linux, com processador AMD *Ryzen* 7 5700U e 13GiB de memória RAM DDR4 disponível. Para emular as redes implantadas foi utilizado a plataforma Mininet.

5.1 AVALIAÇÃO DA TRADUÇÃO DE INTENÇÕES

O primeiro experimento avalia o módulo de tradução de intenções. Em particular, sua função de gerar, a partir de uma descrição em linguagem natural (*i.e.,* a intenção), o PPT. As métricas capturadas neste experimento incluem:

- O tempo em milissegundos necessário para o sistema processar a entrada em linguagem natural e gerar o arquivo JSON;
- A corretude da estrutura do PPT gerado, *i.e.,* se o arquivo está em conformidade sintática à especificação detalhada na Seção 4.2.1;
- A acurácia semântica do arquivo PPT gerado, *i.e.,* o grau em que o JSON gerado se adequa ao que foi expressado na intenção. Isto é medido através de uma porcentagem, que diminui conforme a quantidade de imprecisões encontradas.

A fim de realizar estas medições, foram utilizadas 40 intenções, divididas igualmente em quatro níveis crescentes de complexidade. No nível 1 encontram-se definições diretas de topologias com no máximo 10 *hosts* e conectividade parametrizada (*e.g.,* “Crie uma topologia estrela com um *switch* central s1 e 8 *hosts* (h1 a h8) conectados a ele. Todas as conexões devem ter 100Mbps de banda.”). No nível 2 as intenções lidam com topologias maiores (*i.e.,* 10-20 *hosts*) e condições para aplicar os parâmetros (*e.g.,* “Topologia *DataCenter*: 2 *switches Core* (s1, s2) interligados. 8 *hosts* no s1 e 8 *hosts* no s2. Todos os *hosts* do s1 devem ter 2048MB de RAM. Os do s2 apenas 512MB.”). Já no nível 3, estão intenções relacionadas a topologias com até 50 *hosts* e condições mais abstratas de parametrização (*e.g.,* “Gostaria de uma rede com um total de 30 *hosts*. Eles devem ser conectados em 3 *switches*, dividindo igualmente entre a quantidade de *switches*. No primeiro grupo, defina uma largura de banda máxima de 256 MB para todas as conexões. No segundo, limite a CPU em 75%. Para o último, as máquinas de número par devem ter sua memória limitada em 100MB.”). Por fim, no nível 4, as topologias podem chegar até 60 *hosts*, além de que as intenções são propositalmente mais abstratas, definindo aspectos implícitos (*e.g.,* “Tenho um orçamento de energia restrito. Crie uma rede com 4 *hosts*. A soma total da RAM alocada para todos os *hosts* não pode ultrapassar 1024MB. Distribua essa memória de forma desigual, onde h1 obtém 50% do total.”).

É importante ressaltar que nestes experimentos buscou-se utilizar intenções que são de fato traduzíveis, ou seja, que estejam dentro do escopo permitido pela IBN. A ideia deste experimento é avaliar a capacidade de tradução dentro de entradas esperadas no contexto desta IBN específica, afinal, a utilização de intenções que abarcassem parâmetros não suportados só

seria adequada em outro contexto, no qual se buscasse medir o comportamento da tradução mediante entradas que não deveriam ser possíveis de serem traduzidas de maneira adequada.

Para medir a acurácia das intenções foi atribuída uma porcentagem a cada uma das traduções, calculada com base na quantidade de erros encontrados na tradução. Cada erro encontrado traduz-se em um decréscimo de 5 pontos percentuais à porcentagem, que inicia em 100%. Pelo fato de não terem sido encontrados graus diversos nos erros, isto é, todos os erros foram pequenos, julgou-se desnecessário atribuir descontos diferentes por erro. Os valores absolutos obtidos estão presentes na Tabela 5.2 do Apêndice A.

Tabela 5.1: Tempo de Resposta (ms) da Tradução por Nível de Complexidade

Nível	**Amostras**	**Média**	**Mínimo**	**Máximo**	**Desvio Padrão**
1 (Simples)	10	5000.71	3888.63	6992.95	1010.20
2 (Moderada)	10	9450.32	5662.62	13378.49	2745.83
3 (Complexa)	10	13560.38	6672.55	20315.76	4252.58
4 (Alta Complexidade)	10	9024.66	4349.62	16109.13	4415.97

A expectativa inicial era de que os indicadores de acurácia diminuíssem conforme o aumento na complexidade das intenções, ao passo em que o tempo de tradução também aumentasse de maneira proporcional. Os resultados apresentados na Tabela 5.1 indicam este aumento progressivo de tempo, havendo, no entanto, uma pequena diminuição do nível 3 ao 4. Por conta da natureza "*black box*" das LLMs, é difícil apontar exatamente para a causa disto, no entanto, o mais provável é que o principal fator afetando o tempo de tradução seja o tamanho da topologia gerada, afinal, as topologias do nível 3 são as que apresentam em média o maior número de componentes (*i.e., hosts* ou *switches*).

Outro fator que pode aumentar o tempo é a definição implícita de certos aspectos da intenção, já que nestes casos não há uma instrução clara para a LLM processar. Ela deve, portanto, tomar decisões próprias ou realizar cálculos a fim de traduzir completamente a entrada. Isto verifica-se pois em intenções que apresentam estes aspectos (*e.g.,* "Preciso de uma rede para suportar 45 funcionários. Utilize a quantidade mínima necessária de *switches*, sabendo que cada *switch* suporta apenas 16 conexões. Conecte os *switches* em linha."), houve um tempo de execução mais significativo.

Tabela 5.2: Distribuição da Acurácia das Intenções por Nível de Complexidade

Classificação	**Nível 1**	**Nível 2**	**Nível 3**	**Nível 4**	**Total Geral**
Totalmente Satisfeita (100%)	1	1	1	2	**5**
Satisfeita (75–99%)	9	8	9	5	**31**
Parcialmente Satisfeita (50–74%)	0	1	0	3	**4**
Parcialmente Insatisfeita (25–49%)	0	0	0	0	**0**
Insatisfeita (1–24%)	0	0	0	0	**0**
Totalmente Insatisfeita (0%)	0	0	0	0	**0**
Média de Erros por Nível	**1.0**	**1.4**	**1.2**	**3.1**	–

Com relação à acurácia, os valores mantiveram-se similares ao longo dos três primeiros níveis, havendo uma queda maior apenas no nível 4. Considerando a capacidade dos modelos LLM em gerar respostas altamente complexas, era esperado que arquivos JSON estruturados

não fossem um grande desafio. O grande problema, no entanto, são os casos de alucinação, que acontecem, sobretudo, em entradas pouco claras – como as mencionadas no parágrafo anterior –, e que resultam em arquivos PPT com a configuração de parâmetros que não haviam sido solicitados.

Um exemplo do caso citado acima ocorre ao expressar a seguinte intenção: "Crie uma rede com 12 *hosts* conectados ao *switch* s1. Os primeiros 6 *hosts* são '*Legacy*' e devem ter links de 10Mbps. Os últimos 6 são 'Modern' com links de 1Gbps.". Não houve nenhuma menção sobre limitação de memória ou do uso de CPU para nenhum *host*. No entanto, estes parâmetros acabaram sendo preenchidos. Este é um aspecto inevitável das IAs generativas, mas que poderia ser atenuado por uma engenharia de *prompt* robusta.

Estes valores, no geral, são bastante positivos, e indicam que a LLM é capaz de traduzir eficazmente as traduções. No entanto, deve-se levar em conta que este protótipo possui um escopo ainda limitado de intenções possíveis de serem expressas. Em um ambiente menos controlado, com uma implementação que permitisse a implantação de topologias com parâmetros mais complexos e maiores, espera-se que a capacidade de tradução seja reduzida. Isto seria esperado já que, com mais possibilidades de expressão, aumenta-se a quantidade de informações de entrada da LLM, *i.e.,* o *prompt* fica cada vez mais extenso. Ademais, este aumento de informações de entrada acarretaria em um tempo maior de execução, mesmo para topologias simples, conforme explicado anteriormente.

5.2 AVALIAÇÃO DO MÓDULO DE RECUPERAÇÃO

O segundo experimento tem como objetivo avaliar o mecanismo de recuperação implementado, violando intencionalmente as intenções referentes à perda de conectividade. Para tal, erros foram injetados em dispositivos da rede, o que tornou possível testar a capacidade de verificação e garantia da conectividade entre *hosts* em topologias lineares com um número crescente de *switches*. Ao longo do experimento, foram capturadas as seguintes métricas:

- Tempo em segundos para detectar o erro depois da injeção;
- Tempo em segundos para a rede se recuperar do erro depois da detecção.

O experimento consistiu em realizar a implantação de uma rede, cuja topologia consiste de 2 *hosts* interconectados por um número variável de *switches*, bloquear a conexão entre os *hosts* e medir quanto tempo o sistema demora para detectar a queda da conexão e recuperar-se dela. Para a realização dessas medições, o teste foi executado cinco vezes para cinco tamanhos de topologia distintos, *i.e.,* para 1, 10, 20, 50 e 100 *switches* intermediários entre os dois *hosts*. A fim de bloquear a conexão entre os *hosts*, a interface de rede de um dos *switches* era desabilitada com o comando `ip link set {iface} down`.

Inicialmente, era esperado que o tempo de detecção e recuperação do erro aumentasse com a adição de novos *switches* na topologia. A Figura 5.1 apresenta os resultados das medições da detecção e a Figura 5.2 apresenta os resultados das medições da recuperação. Observa-se que não houveram picos de latência significativos na detecção, e que os valores se mantiveram constantemente entre 0 e 5, distribuindo-se quase que igualmente ao longo deste intervalo.

Nota-se, ao observar as médias dos gráficos, tanto o de detecção quanto o de recuperação, que o impacto da quantidade de *switches* foi mínimo. Isto leva a crer que este fator não seja determinante para o desempenho dos módulos de recuperação e detecção. É possível que surja uma diferença para um número ainda mais elevado de *switches*, no entanto, topologias com

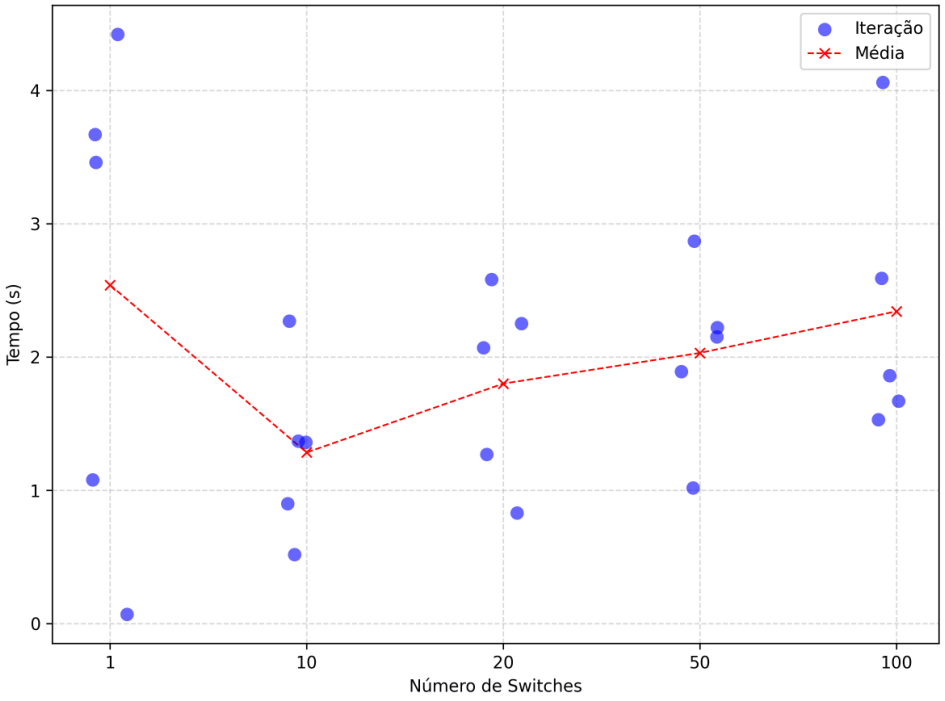


Figura 5.1: Gráfico do tempo de detecção de erro

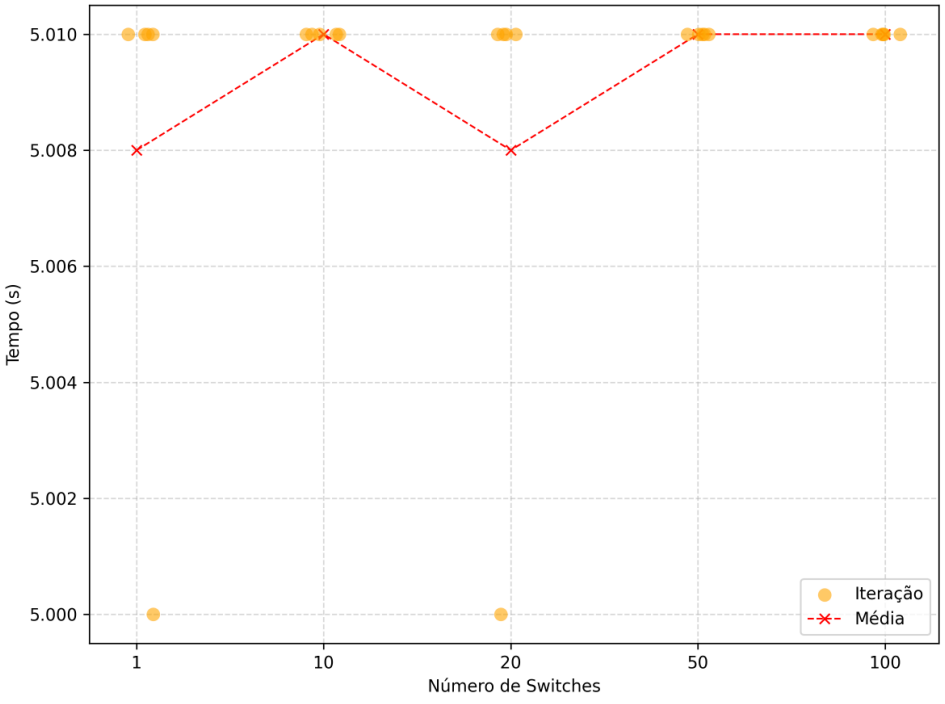


Figura 5.2: Gráfico do tempo de recuperação de erro

mais de 100 *switches* demoram muito para serem colocadas em funcionamento na Mininet, tornando-as inviáveis de serem avaliadas na prática.

O aspecto mais determinante para estes resultados é que os mecanismos de detecção e recuperação funcionam em um ciclo, ou seja, em um intervalo especificado. Neste ciclo, todas as intenções são verificadas, e, caso haja alguma violação, os mecanismos de recuperação são

ativados logo em seguida. Especificamente para estes experimentos, a conectividade entre os *hosts* era verificada a cada 5 segundos. Esta particularidade impõe, na prática, um piso para o tempo de recuperação, já que a confirmação do sucesso da recuperação só ocorre no ciclo de monitoramento posterior àquele no qual houve a detecção da quebra de intenção. Já em relação à detecção, este comportamento periódico impõe uma espécie de teto, afinal, o tempo entre a introdução da falha e a verificação nunca será maior que a duração do ciclo, salvo em casos nos quais a duração da verificação em si excede a duração do ciclo (*e.g.,* se o ciclo dura 3 segundos, mas demorar 5 segundos para verificar todas as intenções).

Outro ponto a ser considerado é que o fator determinante para o tempo de recuperação e detecção dos erros muda dependendo da arquitetura da solução proposta e dos mecanismos implementados. Pelos resultados avaliados, fica claro que o número de *switches* em uma rede, ao menos para redes com até 100 *switches*, não é um fator determinante na implementação proposta neste trabalho, mas supõe-se que o número de *hosts* seja. Isto se dá pelo fato de que as operações de verificação de intenção possuem um *overhead* significativo. No caso da verificação de conectividade, por exemplo, utiliza-se o comando `ping`, que, quando realizado para muitos *hosts* pode acarretar em uma operação de duração considerável.

Vale considerar também que, no protótipo utilizado, a verificação é linear e central. Ou seja, cada intenção é verificada uma após a outra e sempre pelo mesmo processo. Como o *overhead* do aumento do número de *switches* mostrou-se pequeno, isto não foi um problema de desempenho. No entanto, poderia ser caso a quantidade de *hosts* estivesse aumentando. Uma alternativa seria a paralelização do processo de verificação, que poderia evitar, ou ao menos mitigar, o gargalo causado por realizar todas as checagens de forma centralizada.

6 CONCLUSÃO

O paradigma de redes IBN é ainda incipiente e, para que ele seja amplamente adotado, ainda existem diversos desafios que precisam ser superados. Neste sentido, este trabalho propôs a PIÁ, uma arquitetura para redes IBN orientada a *plugins*. A arquitetura PIÁ especifica um arquivo, denominado de PPT, que contém as definições do usuário, explicitando a topologia da rede de acordo com as intenções sobre ela. A PPT pode ser gerada a partir de linguagem natural pelo *LLM Translator*. Recebendo a PPT, há um sistema dividido em quatro componentes: *Parsing*, *Deployment*, *Monitoring* e *Assurance*. Este sistema, por sua vez, pode receber *plugins*, projetados para ampliar a funcionalidade geral de acordo com as necessidades específicas de cada contexto.

Um protótipo da arquitetura PIÁ foi implementado e avaliado. Em particular, dois componentes considerados relevantes para o sistema foram testados – a tradução da intenção e a garantia dela através do monitoramento e da recuperação. Os experimentos realizados apontam para um sucesso significativo no que diz respeito à utilização de uma LLM para traduzir as intenções em linguagem natural para definições estruturadas de uma topologia parametrizada. Isto se verifica pela alta taxa de acurácia das traduções testadas, e pela baixa ocorrência de alucinações. Há ainda de se considerar a constante melhora dos modelos de LLM que, juntos de uma engenharia de *prompt* robusta, podem estabelecer um novo método consolidado para realizar a operação de tradução de intenções em LLM.

Já em relação à garantia de intenções, constatou-se a capacidade do protótipo de lidar com topologias lineares com um número considerável de *switches*. Por outro lado, os mecanismos de recuperação implementados podem encontrar dificuldades em outras topologias (*e.g.,* que possuem um número elevado de *hosts*). Além disso, os resultados obtidos a partir desta implementação específica não necessariamente serão encontrados em outros casos, já que pode haver uma mudança significativa no funcionamento destes mecanismos dependendo do contexto da rede.

De modo geral, todos estes resultados indicam um avanço no campo da IBN, que pode ser ainda mais significativo em trabalhos futuros. Em particular, destaca-se a possibilidade de ampliar o desenvolvimento do módulo de tradução, buscando avaliar o comportamento das traduções diante de intenções mais diversas (*e.g.,* em casos de intenções que versam sobre *plugins* ou que solicitem aspectos inviáveis para a rede), além da comparação de diferentes modelos LLM do estado da arte.

Ainda, há a possibilidade de desenvolver uma implementação ainda mais robusta e fora de um ambiente simulado como o da Mininet, verificando a capacidade de integração da arquitetura em ambientes de produção.

REFERÊNCIAS

Chaudhari, A., Asthana, A., Kaluskar, A., Gedia, D., Karani, L., Perigo, L., Gandotra, R. e Gangwar, S. (2019). VIVoNet: Visually-Represented, Intent-Based, Voice-Assisted Networking. *International Journal of Computer Networks & Communications*, 11(02):01–13.

Chiosi, M., Clarke, D., Willis, P., Reid, A., Feger, J., Bugenhagen, M., Khan, W., Fargano, M., Cui, C., Deng, H. et al. (2012). Network functions virtualisation: An introduction, benefits, enablers, challenges and call for action. Em *SDN and OpenFlow World Congress*, páginas 22–24.

Claise, B., Quilbeuf, J., Lopez, D., Voyer, D. e Arumugam, T. (2023). Service Assurance for Intent-Based Networking Architecture. Relatório técnico, RFC Editor.

Clemm, A., Ciavaglia, L., Z. Granville, L. e Tantsura, J. (2022). Intent-Based Networking - Concepts and Definitions. Relatório Técnico RFC9315, RFC Editor.

Cohen, R., Barabash, K., Rochwerger, B., Schour, L., Crisan, D., Birke, R., Minkenberg, C., Gusat, M., Recio, R. e Jain, V. (2013). An intent-based approach for network virtualization. Em *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, páginas 42–50. ISSN: 1573-0077.

Costa, P., Migliavacca, M., Pietzuch, P. e Wolf, A. L. (2012). NaaS: Network-as-a-Service in the cloud. Em *2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE '12)*, San Jose, CA. USENIX Association.

Edeline, K. e Donnet, B. (2019). A Bottom-Up Investigation of the Transport-Layer Ossification. Em *2019 Network Traffic Measurement and Analysis Conference (TMA)*, páginas 169–176.

Faraci, G. e Schembra, G. (2015). An Analytical Model to Design and Manage a Green SDN/NFV CPE Node. *IEEE Transactions on Network and Service Management*, 12(3):435–450.

Gupta, P. (2025). Intent-Based Networking Architecture: A Deep Dive into Its Components and Workflow. *Journal of Computer Science and Technology Studies*, 7:586–597.

Imran, H. A., Latif, U., Ikram, A. A., Ehsan, M., Ikram, A. J., Khan, W. A. e Wazir, S. (2020). Multi-Cloud: A Comprehensive Review. Em *2020 IEEE 23rd International Multitopic Conference (INMIC)*, páginas 1–5. ISSN: 2049-3630.

Jacobs, A., Pfitscher, R., Ferreira, R. e Granville, L. (2018). Refining Network Intents for Self-Driving Networks. páginas 15–21.

Jacobs, A. S., Pfitscher, R. J. e Ribeiro, R. H. (2021). Hey, Lumi! Using Natural Language for Intent-Based Network Management.

Kephart, J. e Chess, D. (2003). The Vision Of Autonomic Computing. *Computer*, 36(1):41–50.

Kreutz, D., Ramos, F. M. V., Veríssimo, P. E., Rothenberg, C. E., Azodolmolky, S. e Uhlig, S. (2015). Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*, 103(1):14–76.

Kulkarni, G. (2017). Simplification of Internet Ossification through Software Defined Network Approach. *International Journal on Future Revolution in Computer Science Communication Engineering*.

Leivadeas, A. e Falkner, M. (2023a). Autonomous Network Assurance in Intent Based Networking: Vision and Challenges. Em *2023 32nd International Conference on Computer Communications and Networks (ICCCN)*, páginas 1–10. ISSN: 2637-9430.

Leivadeas, A. e Falkner, M. (2023b). A Survey on Intent-Based Networking. *IEEE Communications Surveys & Tutorials*, 25(1):625–655.

Li, C., Havel, O., Olariu, A., Martinez-Julia, P., Nobre, J. e Lopez, D. (2022). Intent Classification. Relatório técnico, RFC Editor.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S. e Turner, J. (2008). Openflow: Enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74.

Mijumbi, R., Serrat, J., Gorricho, J.-L., Bouten, N., De Turck, F. e Boutaba, R. (2016). Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 18(1):236–262.

Minhas, S., Jaswal, R., Sharma, A. e Singla, S. (2024). Revolutionizing Networking: A Comprehensive Overview of Intent-Based Networking. Em *2024 International Conference on Emerging Innovations and Advanced Computing (INNOCOMP)*, páginas 463–468.

Njah, Y., Leivadeas, A. e Falkner, M. (2025). An AI-Driven Intent-Based Network Architecture. *IEEE Communications Magazine*, 63(4):146–153.

ONAP (2025). Onap. Acessado em novembro de 2025.

Pang, L., Yang, C., Chen, D., Song, Y. e Guizani, M. (2020). A Survey on Intent-Driven Networks. *IEEE Access*, 8:22862–22873.

Papastergiou, G., Fairhurst, G., Ros, D., Brunstrom, A., Grinnemo, K.-J., Hurtig, P., Khademi, N., Tuxen, M., Welzl, M., Damjanovic, D. e Mangiante, S. (2017). De-Ossifying the Internet Transport Layer: A Survey and Future Perspectives. *IEEE Communications Surveys & Tutorials*, 19(1):619–639. Publisher: Institute of Electrical and Electronics Engineers (IEEE).

Raghavan, B., Casado, M., Koponen, T., Ratnasamy, S., Ghodsi, A. e Shenker, S. (2012). Software-defined internet architecture: decoupling architecture from infrastructure. Em *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, páginas 43–48, Redmond Washington. ACM.

Santana, B. S., Gaiardo, G. d. F., Marcuzzo, L., Lucca, L. P. e Tavares, T. N. (2017). JMP: Uma Solução para Definição de Topologias Mininet Utilizando JSON.

Saraiva, N., Islam, N., Lachos Perez, D. A. e Esteve Rothenberg, C. (2019). Policy-Driven Network Traffic Rerouting Through Intent-Based Control Loops. Em *Anais do Workshop de Gerência e Operação de Redes e Serviços*, páginas 15–28. Sociedade Brasileira de Computação - SBC.

Singh, A. (2025). Intent-based networking in multi-cloud environments. *Journal of Engineering and Applied Sciences Technology*, 7(2):1–7.

Sullivan, N. (2017). Why TLS 1.3 isn't in browsers yet.

Team, G. et al. (2023). Gemini: A family of highly capable multimodal models.

Toy, M. (2021). Intent-based networking for connectivity and cloud services. *Advances in Networks*, 9(1):19–22.

Wang, J., Zhang, L., Yang, Y., Zhuang, Z., Qi, Q., Sun, H., Lu, L., Feng, J. e Liao, J. (2023). Network Meets ChatGPT: Intent Autonomous Management, Control and Operation. *Journal of Communications and Information Networks*, 8(3):239–255.

Yang, H., Zhan, K., Yao, Q., Zhao, X., Zhang, J. e Lee, Y. (2020). Intent defined optical network with artificial intelligence-based automated operation and maintenance. *Science China Information Sciences*, 63(6):160304.

Yu, A., Yang, H., Yao, Q., Li, Y., Guo, H., Peng, T., Li, H. e Zhang, J. (2019). Accurate Fault Location Using Deep Belief Network for Optical Fronthaul Networks in 5G and Beyond. *IEEE Access*, 7:77932–77943.

Yu, H., Rahimi, H., Janz, C., Wang, D., Li, Z., Yang, C. e Zhao, Y. (2024). Building a Comprehensive Intent-Based Networking Framework: A Practical Approach from Design Concepts to Implementation. *Journal of Network and Systems Management*, 32(3):47.

Zeydan, E. e Turk, Y. (2020). Recent Advances in Intent-Based Networking: A Survey. páginas 1–5.

APÊNDICE A – TABELAS

Tabela A.1: Entradas e Métricas Completas do Experimento 1 (Parte 1/4)

ID	Nivel	Prompt	Tempo (ms)	JSON Válido	Acuracia Semântica	Obs
1	1	Crie uma topologia estrela com um switch central s1 e 8 hosts (h1 a h8) conectados a ele. Todas as conexões devem ter 100Mbps de banda.	5389.34	Sim	95%	Definiu desnecessariamente os IPs
2	1	Gere uma rede linear com 3 switches (s1, s2, s3) conectados em série. Conecte 2 hosts em cada switch. Defina o delay de todos os links para 10ms.	6242.2	Sim	95%	Definiu desnecessariamente os IPs
3	1	Preciso de uma rede com 10 hosts conectados ao switch s1. Os hosts h1 a h5 devem ter o IP na sub-rede 10.0.1.0/24 e os hosts h6 a h10 na sub-rede 10.0.2.0/24.	5513.31	Sim	100%	-
4	1	Configure uma rede com 6 hosts e 1 switch. Habilite o monitoramento com intervalo de 5 segundos e recuperação automática para garantir conectividade.	4343.49	Sim	95%	Definiu desnecessariamente os IPs
5	1	Crie uma rede com 5 hosts. O host h1 é o servidor e deve ter MAX_CPU de 1.0 (100%) e 1024MB de RAM. Os outros (h2-h5) devem ter CPU limitada a 20%.	6992.95	Sim	90%	Definiu desnecessariamente os IPs e os endereços MAC
6	1	Topologia com dois switches, s1 e s2. Conecte h1, h2, h3 em s1 e h4, h5, h6 em s2. O link entre os switches deve ser de alta velocidade (1000Mbps).	4264.63	Sim	95%	Definiu desnecessariamente os IPs
7	1	Gere uma rede com 8 hosts ligados ao s1. Todos os links devem ter 1% de perda de pacotes simulada para teste de robustez.	4159.22	Sim	95%	Definiu desnecessariamente os IPs
8	1	Conecte 4 hosts ao s1. Configure um controlador remoto no IP 192.168.56.1, porta 6633. Todos os hosts devem ter 512MB de RAM.	3880.63	Sim	95%	Definiu desnecessariamente os IPs
9	1	Crie uma rede em anel com 4 switches (s1-s2-s3-s4-s1). Conecte um host em cada switch. Defina a banda dos links entre switches como 500Mbps.	4710.4	Sim	95%	Definiu desnecessariamente os IPs
10	1	Rede simples com 10 hosts no switch s1. Porém, quero que o monitoramento esteja ativado apenas para reportar falhas, sem tentar recuperar (recovery_enabled false).	4510.98	Sim	95%	Definiu desnecessariamente os IPs

Tabela A.2: Entradas e Métricas Completas do Experimento 1 (Parte 2/4)

ID	Nivel	Prompt	Tempo (ms)	JSON Válido	Acuracia Semântica	Obs
11	2	Crie uma rede com 12 hosts conectados ao switch s1. Os primeiros 6 hosts são 'Legacy' e devem ter links de 10Mbps. Os últimos 6 são 'Modern' com links de 1Gbps.	11253.22	Sim	70%	Definiu desnecessariamente os IPs e os endereços MAC. Também colocu alguns parâmetros de host aleatórios que não foram solicitados: 2x RAM e 2x CPU
12	2	Gere uma topologia com 15 hosts divididos em 3 switches (5 por switch). O Switch 1 é o Core e seus hosts devem ter prioridade de CPU (80%). Os outros não têm limite.	13378.49	Sim	95%	Definiu desnecessariamente os IPs
13	2	Topologia DataCenter: 2 switches Core (s1, s2) interligados. 8 hosts no s1 e 8 hosts no s2. Todos os hosts do s1 devem ter 2048MB de RAM. Os do s2 apenas 512MB.	7418.01	Sim	95%	Definiu desnecessariamente os IPs
14	2	Crie uma rede com 20 hosts em um único switch. Os hosts com ID de h1 a h10 devem ter atraso de 5ms. Os hosts de h11 a h20 devem ter atraso de 50ms.	8114.73	Sim	95%	Definiu desnecessariamente os IPs
15	2	Sistema de Vigilância: 10 câmeras (h1-h10) e 1 servidor (h11) no switch s1. As câmeras têm banda limitada a 5Mbps. O servidor tem banda de 1000Mbps e CPU livre.	7380.54	Sim	95%	Definiu desnecessariamente os IPs
16	2	Rede com 16 hosts. Divida-os em 4 switches (s1, s2, s3, s4). Ative o monitoramento com intervalo agressivo (2s). Apenas os hosts do s1 precisam de limite de memória (256MB).	12487.32	Sim	95%	Definiu desnecessariamente os IPs. Também não colocou um link entre os switches, apesar de que isso realmente não foi solicitado
17	2	Conecte 10 hosts ao s1. Configure IPs sequenciais começando de 192.168.0.10 até 192.168.0.19. O link de todos deve ter 0.5% de perda.	5662.62	Sim	100%	-
18	2	Simulação VoIP: 14 hosts no switch s1. Hosts pares devem ter configuração de Jitter simulado (Delay 20ms). Hosts ímpares devem ter conexão perfeita (Delay 1ms).	8054.54	Sim	95%	Definiu desnecessariamente os IPs
19	2	Rede de 18 hosts distribuídos em 2 switches. O link entre os switches deve ser gargalo (10Mbps). Os links dos hosts devem ser rápidos (100Mbps). Monitoramento ligado.	12799.91	Sim	95%	Definiu desnecessariamente os IPs
20	2	Crie uma rede com 12 hosts. Hosts h1, h5 e h9 são gateways e precisam de MACs fixos (00:00:00:00:00:01, etc). O restante pode ser automático.	7953.86	Sim	95%	Definiu desnecessariamente os IPs

Tabela A.3: Entradas e Métricas Completas do Experimento 1 (Parte 3/4)

ID	Nivel	Prompt	Tempo (ms)	JSON Válido	Acuracia Semântica	Obs
21	3	Gostaria de uma rede com um total de 30 hosts. Eles devem ser conectados em 3 switches, dividindo igualmente entre a quantidade de switches. No primeiro grupo, defina uma largura de banda maxima de 256 MB para todas as conexoes. No segundo, limite o cpu em 75%. Para o ultimo, as maquinas de numrero par devem ter sua memoria limitada em 100MB.	20215.76	Sim	95%	Definiu desnecessariamente os IPs
22	3	Crie uma simulação de escritório com 40 hosts divididos em 4 departamentos (4 switches). Departamento A (h1-h10) precisa de alta CPU (90%). Departamento B (h11-h20) precisa de alta RAM (1024MB). Departamentos C e D são padrão. Todos conectados a um switch central s_core.	17393.92	Sim	95%	Definiu desnecessariamente os IPs
23	3	Gere uma rede massiva com 50 hosts conectados a um switch s1. A cada 10 hosts, o link deve ter uma degradação diferente: 1-10 (0% loss), 11-20 (1% loss), 21-30 (2% loss), e assim por diante.	14929.49	Sim	95%	Definiu desnecessariamente os IPs
24	3	Topologia em Árvore Binária: Switch Raiz conecta a 2 switches, que conectam a mais 2 cada (Total 7 switches). Coloque 3 hosts em cada switch da ponta (folhas). Total de 12 hosts. Monitoramento com recuperação ativado.	6672.55	Sim	95%	Definiu desnecessariamente os IPs
25	3	Rede com 30 hosts. Os hosts com ID múltiplo de 5 (h5, h10, etc.) são servidores críticos: CPU 100%, RAM 2048MB e Link 1Gbps. O restante são clientes com CPU 20% e Link 10Mbps.	16235	Sim	90%	Definiu desnecessariamente os IPs e os endereços MAC
26	3	Crie 3 grupos de 8 hosts (Total 24). Grupo 1 (Switch 1) é 'Voz' (Delay 2ms). Grupo 2 (Switch 2) é 'Dados' (Banda 1000Mbps). Grupo 3 (Switch 3) é 'IoT' (Banda 1Mbps). Conecte os 3 switches em anel.	10376.3	Sim	95%	Definiu desnecessariamente os IPs
27	3	Cenário de Falha em Cascata: 30 hosts, 3 switches em cadeia. O switch do meio (s2) tem 10 hosts e deve ter seus links configurados com 50% de perda de pacotes (simulando falha). Os switches das pontas (s1, s3) funcionam normal.	14685.57	Sim	85%	Definiu desnecessariamente os IPs e colocou loss entre os switches. Me parece que só deveria haver loss entre os hosts e o switch, não entre os switches, mas ficou ambíguo
28	3	Desafio de endereçamento: 25 hosts no switch s1. O IP deve seguir a regra: 10.0.X.1, onde X é o número do host (ex: h1 = 10.0.1.1, h25 = 10.0.25.1). Limite a CPU de todos em 50%.	10802.47	Sim	100%	-
29	3	Rede com 32 hosts divididos em 2 switches. No primeiro switch, todos os hosts devem ter MAC terminando em número par. No segundo switch, MAC terminando em ímpar. Monitoramento a cada 10s.	15631.38	Sim	95%	Definiu desnecessariamente os IPs
30	3	Crie uma rede com 20 hosts onde a largura de banda decresce. h1 tem 100Mbps, h2 tem 95Mbps, h3 tem 90Mbps... até h20. Conectados ao s1.	8661.37	Sim	95%	Definiu desnecessariamente os IPs

Tabela A.4: Entradas e Métricas Completas do Experimento 1 (Parte 4/4)

ID	Nivel	Prompt	Tempo (ms)	JSON Válido	Acuracia Semântica	Obs
31	4	Preciso de uma rede para suportar 45 funcionários. Utilize a quantidade mínima necessária de switches, sabendo que cada switch suporta apenas 16 conexões. Conecte os switches em linha.	15942.37	Sim	60%	Definiu desnecessariamente os IPs, e colocou um switch a mais do que o necessário, sem conectar o máximo de hosts por switch
32	4	Crie uma rede com 10 hosts no switch s1. Configure todos os links com 10Mbps e 50ms de delay, EXCETO os hosts h1 e h2, que devem ter conexão 'ideal' (1Gbps, 1ms).	5878.01	Sim	95%	Definiu desnecessariamente os IPs
33	4	Conecte 5 hosts ao s1. O host h1 rodará uma aplicação de Cirurgia Remota (requer latência ultra-baixa e zero perda). Os outros são para navegação web comum. Defina os parâmetros técnicos apropriados para cada caso.	6280.52	Sim	90%	Configurou o MAC do H1 e limitou seu cpu e ram
34	4	Tenho 20 hosts em um switch. Se o ID do host for primo, ele é um servidor seguro (CPU 90%). Se não for primo, mas for par, é uma estação de trabalho (CPU 20%). Os ímpares não primos não devem ter limite de CPU.	11401.41	Sim	95%	Definiu desnecessariamente os IPs
35	4	Tenho um orçamento de energia restrito. Crie uma rede com 4 hosts. A soma total da RAM alocada para todos os hosts não pode ultrapassar 1024MB. Distribua essa memória de forma desigual, onde h1 pega 50% do total.	5548.99	Sim	55%	Definiu desnecessariamente os IPs e os MACs, colocou 50% no primeiro, como esperado, mas não distribuiu o resto igualmente, e colocou restrições de largura de banda sem que isso fosse mencionado
36	4	Simule a rede de uma pequena empresa: 1 CEO, 2 Gerentes e 10 Estagiários. O CEO precisa de conexão direta exclusiva a um switch s_vip. Os gerentes ficam no s_mgmt. Os estagiários no s_staff. Interligue os switches.	11167.67	Sim	60%	Definiu denecessariamente os IPs e os MACs, e colocou restrições de cpu e ram no host ceo e nos hosts managers.
37	4	Conecte 60 hosts ao s1. Configure os IPs usando a sub-rede 192.168.0.0/26. Garanta que todos tenham IPs únicos dentro desse range válido.	16109.13	Sim	100%	-
38	4	Crie uma topologia Malha Total (Full Mesh) entre 5 switches (s1 a s5), onde cada switch tem 2 hosts. Habilite a recuperação no monitoramento apenas se a redundância física falhar.	8489.87	Sim	95%	Definiu desnecessariamente os IPs
39	4	Crie uma rede com 8 hosts. Simule um ataque DDoS no h1 vindo do h8: o link do h1 deve ter uso de CPU em 100% e o link do h8 deve ter largura de banda consumida (configure banda minúscula de 1kbps).	5079.04	Sim	95%	Definiu desnecessariamente os IPs
40	4	Crie uma cadeia de 5 switches (s1 a s5). Um host em cada ponta (h_start em s1, h_end em s5). A latência deve acumular: 10ms no primeiro link entre switches, dobrando a cada salto subsequente (10, 20, 40, 80).	4349.62	Sim	100%	-