UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

LEVINDO GABRIEL TASCHETTO NETO

PLACIDUS: Plataforma de Gerenciamento com Verificação Formal para Redes Definidas por Software

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Engenharia da Computação

Orientador: Prof. Dr. Alberto Egon Schaeffer-Filho
Co-orientador: Me. Anderson Santos da Silva

Porto Alegre
2020

“Information technology and business are becoming inextricably interwoven. I don’t think anybody can talk meaningfully about one without the talking about the other.”

— Bill Gates

AGRADECIMENTOS

Agradeço à Universidade Federal do Rio grande do Sul por ter me dado a oportunidade de estudar e me desenvolver como aluno e profissional dentro de um excelente ambiente de ensino, em especial ao professor Alberto Egon Schaeffer-Filho pela orientação nesse trabalho. Também agradeço à Universidade de Stuttgart por ter me recebido durante o ano 2017/18, em especial ao meu orientador durante o período, professor Bernhard Mitschang.

Além disso agradeço infinitamente à minha mãe, Leonice, e ao meu pai, Edir, o qual partiu no início da minha graduação. Ademais, um agradecimento à minha namorada, Brenda, por todo apoio, paciência e incentivo. Também, a todos meus amigos e familiares que me ajudaram, de alguma forma, durante esses anos de bacharelado em Engenharia de Computação na UFRGS.

Agradeço também ao Grupo de Redes do Instituto de Informática da UFRGS e colegas do laboratório 212, especialmente ao meu co-orientador Anderson Santos da Silva, o qual me ajudou muito na construção desse trabalho desde seu início de pesquisa e desenvolvimento. Por fim, agradeço imensamente a Deus, por ter me dado saúde, inteligência e força durante todos esses anos de estudo e dedicação.

RESUMO

Verificação formal é um passo importante na checagem de operações e propriedades em redes de computadores, como por exemplo, a corretude das configurações de dispositivos inseridos na mesma. Entretanto, historicamente, técnicas de verificação formal são limitadas por desempenho, escalabilidade e expressividade, principalmente devido à complexidade das redes de computadores atuais. Problemas para processar uma tarefa ou função específica em uma rede podem ser críticos para a confiabilidade de um sistema como um todo. Além disso, a execução de testes ficam limitadas, na maioria das ocasiões, ao ambiente final de produção. Com validações feitas apenas em estágios mais avançados, quaisquer modificações necessárias ao sistema se tornam caras, e muitas vezes, demoradas. Para tratar essa combinação de problemas, várias técnicas para verificação formal podem ser utilizadas em estágio inicial de desenvolvimento. Essas técnicas podem ser (i) atualização das informações sobre os contextos e estados de dispositivos da rede e (ii) utilização de estratégicas inteligentes para combinar modelos de representação para garantir performance e escalabilidade. Ademais, é preciso um ambiente para simulações de rede para que essas regras sejam verificadas, o qual contenha um emulador de rede e um controlador de rede para obtenção dos fluxos a serem validados dentro da topologia. Arquiteturas para SDN oferecem a essência necessária para resolver problemas desse tipo, uma vez que disponibilizam a combinação de diferentes técnicas para aumentar a qualidade dos resultados obtidos por um sistema de seleção. Nesse trabalho, é proposto o Placidus, que é uma plataforma de gerenciamento com verificação formal para SDN. A plataforma possui incluso um sistema de autenticação de usuários. Ademais, ela tem um dashboard, onde é possível fazer o gerenciamento de topologias de rede, além de ser possível executar a verificação de propriedades das mesmas. A plataforma utiliza um framework flexível, que atualmente possui scripts de verificação para verificar alcançabilidade entre dispositivos de rede, além de inconsistências entre fluxos, tais como redundâncias e conflitos de regras.

Palavras-chave: Redes definidas por software. gerenciamento de rede. protocolo openflow. verificação formal.

PLACIDUS: A Platform of Management with Formal Verification for Software Defined Networking

ABSTRACT

Formal verification is an important step in checking network operations and ensuring properties, for instance, the accuracy of device configurations. Nevertheless, historically, formal verification techniques are limited by performance, scalability, and expressiveness, mainly due to the complexity of the current networks. Problems on processing a specific task or function in computer networks may be critical for the reliability of a whole system. Furthermore, the executions of tests are mostly limited to the final production environment. With variations made only in advanced stages, any needed changes to the system become expensive and very time-consuming. In order to solve these possible complications, several techniques for formal verification can be used in early developing stages. These techniques can be (i) updating the information about the context and state of network devices, and (ii) using intelligent strategies for combining representation models for guaranteeing performance and scalability. In addition, an environment for simulations is required in order to have rules verified within the network topology. This environment ought to have at least a network emulator and a network controller to obtain the flows within the topology. Software defined networking (SDN) architectures offer the essence needed for resolving this kind of issue. It enables the selection of a set of different techniques for increasing the quality of the results obtained by a selection system. In this work, we propose Placidus, which is a platform of management with formal verification for software defined networking. The platform has a user authentication system included. Besides, it has a dashboard where network topologies can be created and rules in it can be verified. It uses a flexible framework, which is currently capable of verifying network properties, such as reachability, redundancy in network devices, and other inconsistencies.

Keywords: Software defined Networking, networking management, Openflow protocol, Formal verification.

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
CSS	*Cascading Style Sheets*
CSV	*Comma-Separated Values*
DDoS	*Distributed Denial of Service*
HTML	*Hypertext Markup Language*
IP	*Internet Protocol*
JSON	*JavaScript Object Notation*
LAN	*Local Area Network*
MAC	*Media Access Control*
NOS	*Network Operating System*
REST	*Representational State Transfer*
SDN	*Software Defined Networking*
SSH	*Secure Shell*
TLS	*Transport Layer Security*
TUFU	*Trust-upon-first-use*

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Neste trabalho é apresentado o Placidus, uma plataforma de gerenciamento com verificação formal para redes definidas por software. Esse capítulo é sub-dividido da seguinte forma: a Seção 1.1 apresenta uma contextualização sobre redes de computadores e onde o trabalho está situado; já a Seção 1.2 apresenta a motivação ao problema resolvido por esse trabalho; a Seção 1.3 mostra as principais contribuições da plataforma; e, por fim, a Seção 1.4 descreve o que será visto em cada capítulo seguinte a esse.

1.1 Contextualização

Redes de computadores têm se tornado cada vez mais importantes na vida das pessoas, sendo aderidas em massa por estabelecimentos de diversos setores, universidades, escolas, entre outros. Para tanto, estudos que visam garantir a operação correta da rede têm se popularizado nos últimos anos (da Silva; Schaeffer-Filho, 2019). Ademais, é possível identificar problemas na rede é através de análise de tráfego (Ahmad et al., 2020) e detecção de *malware*[1] (Yeo et al., 2018). Por essa razão, é extremamente importante identificar fluxos, dentro das topologias de rede, que tenham anomalias e possíveis problemas de configuração.

O uso de soluções baseadas em software, em geral, pode introduzir *misconfigurations*[2], e.g.: inconsistências de roteamento, em serviços críticos de redes de computadores (Al-Haj; Tolone, 2017). Um exemplo disso, são os protocolos tais como *BGP (Border Gateway Protocol)* (Gobrial, 1996), que ainda sofrem com problemas de configuração, principalmente devido a enganos causados por operadores de rede (FEAMSTER; BALAKRISHNAN, 2005). Essa realidade expõe a necessidade de estudos voltados à detecção desse tipo de erro e a classificação de sua origem como maliciosa ou ocasional (Dahl et al., 2013).

Além disso, a configuração facilitada de ambientes para simulações e testes de redes de computadores ainda é uma necessidade relevante nesse meio, uma vez que o tempo necessário para montar um ambiente completo para realizar testes reais ainda é bastante custoso e demorado para escalar (Garrich et al., 2020). Estudos, tais como (SETHI; NARAYANA; MALIK, 2013), sugerem que verificação formal possui um conjunto de téc-

[1] Software nocivo destinado a infiltrar-se em um sistema de forma ilícita.
[2] Configuração incorreta de um sistema ou parte dele.

nicas capaz de verificar ambientes de rede e responder essas questões, já que é uma área ampla, incluindo técnicas tais como *model checking*, o qual é um procedimento de verificação formal para saber se um modelo cumpre uma especificação, sendo essa dada por uma propriedade expressada formalmente em termos de uma fórmula lógica temporal e de maneira automática (Koike; Nishizaki, 2013). Outra técnica é a de *symbolic simulation*, que é uma forma de simulação onde execuções possíveis de um sistema são consideradas simultaneamente para o tratamento de diversos tipos de inconsistências, mesmo que em diferentes domínios (Hamaguchi, 2001). Essa técnica é bastante utilizada na validação de circuitos digitais e permite que a análise desses circuitos seja feita sem que haja a necessidade de especificação de testes de consumo de tempo elevado (Bertacco; Olukotun, 2002).

1.2 Motivação

Apesar de o uso de técnicas de verificação formal de maneira isolada ser frequente, conforme evidenciado por trabalhos como (BALL et al., 2014), é notado em (Kang et al., 2013) que a combinação de fórmulas lógicas e técnicas de verificação distintas mas complementares, dentro de uma plataforma virtual para gerenciamento de rede, é essencial para acelerar a construção de topologias e aumentar as chances de um correto funcionamento das mesmas. Existe uma necessidade adicional de que técnicas de verificação formal sejam sofisticadas, utilizando o contexto e estado dos dispositivos de rede, modelos de representação precisos e estratégias de remediação (Mouradian; Blum, 2014).

Arquiteturas baseadas em *Software-Defined Networking* (SDN) oferecem o ferramental necessário para a aplicação de métodos formais para verificação de protocolos e aplicações de rede (Qadir; Hasan, 2015), uma vez que é possível obter dados dos fluxos de rede de maneira simplificada, através de seus planos de controle logicamente centralizados (Markowski; Ryba; Puchała, 2016). SDN fornece as tecnologias necessárias para pesquisadores testarem aplicabilidades para sustentar a tolerância a falhas de soluções de controle de tráfego, além de facilitar questões de acessibilidade de dispositivos de rede (Asadollahi et al., 2017).

Contudo, ainda existem desafios como (i) detectar falha de configuração proveniente da interação humana nesses ambientes, (ii) gerenciar a rede para inserir, modificar e remover propriedades sem desconfigurar seus dispositivos e (iii) coordenar a interação conjunta de aplicações distintas e complexas de modo que uma não interfira no funcio-

namento interno (Sokappadu et al., 2019). Nesse contexto, essa monografia apresenta a Plataforma Placidus, destinada ao gerenciamento de redes definidas por software, combinando técnicas para verificação formal de regras e propriedades de modo a identificar problemas de configuração e garantir a resiliência da rede.

1.3 Contribuições

O propósito principal da plataforma é servir como auxílio a operadores de rede que desejam montar suas topologias e interfaces de rede em um ambiente seguro e de fácil acesso. A plataforma é disponibilizada para todos os usuários de maneira online. Isso é possível, pois a mesma é instalada na nuvem da Microsoft Azure[3] dentro de uma máquina virtual, o que acaba gerando uma diminuição significativa na quantidade de tráfego de rede para os *hosts*[4] provedores (Anderson; Cho, 2017). Além disso, o Placidus possui autenticação com nome de usuário e senha, para garantir que informações dos testes pessoais e configurações dos operadores de rede não sejam compartilhadas sem o consentimento dos mesmos.

Além de poderem ser feitas simulações de topologias de rede, regras podem ser checadas por meio de algoritmos de verificação formal internos ao Placidus. O Placidus é composto por (i) um emulador de rede para gerenciamento das topologias, (ii) um controlador SDN, (iii) scripts de verificação e (iv) uma interface web, a qual faz uso dos componentes desenvolvidos via API. Detalhes de implementação da arquitetura proposta e validações dos algoritmos implementados são discutidos em seções posteriores.

1.4 Estrutura do Documento

Este trabalho está estruturado da seguinte forma: o Capítulo 2 apresenta uma fundamentação teórica e uma revisão bibliográfica referentes ao contexto no qual o trabalho está inserido; o Capítulo 3 apresenta a solução proposta para plataforma em termos de arquitetura, funcionalidades e tecnologias utilizadas; o Capítulo 4 descreve detalhes dos ambientes utilizados e dos testes implementados, assim como os resultados obtidos; o Capítulo 5, por fim, conclui este trabalho apresentando as considerações finais.

[3] https://azure.microsoft.com/

[4] Máquinas conectadas a uma rede de computadores, as quais podem oferecer recursos, serviços e informações aos seus usuários ou outros nodos na rede.

2 FUNDAMENTAÇÃO TEÓRICA E TRABALHOS RELACIONADOS

Neste capítulo, é apresentada uma fundamentação teórica no contexto do qual o trabalho está inserido. Esse capítulo está estruturado da seguinte maneira: A Seção 2.1 apresenta estudos desenvolvidos no âmbito de Redes Definidas por Software com conceitos relacionados ao protocolo OpenFlow; A Seção 2.2, por sua vez, explora técnicas de verificação de regras em redes de computadores; Por fim, a Seção 2.3 aborda trabalhos desenvolvidos em plataformas de gerenciamento de redes.

2.1 Redes Definidas por Software

Redes Definidas por Software (SDN) se tornaram uma das arquiteturas de redes de computadores mais importantes. Essa popularidade de SDN se deve ao fato da mesma permitir a simplificação do gerenciamento de rede, assim como acelerar a inovação em redes de comunicação (FEAMSTER; REXFORD; ZEGURA, 2014).

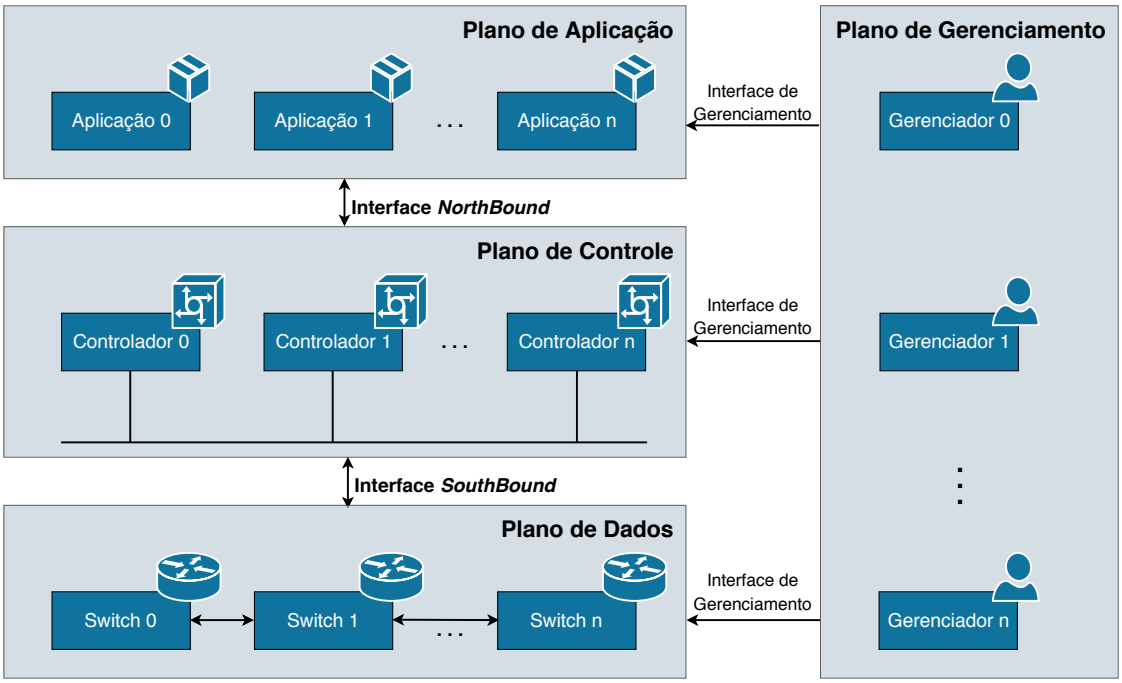
SDN é uma arquitetura proposta para ser facilmente gerenciável, de baixo custo e adaptável, para se adequar a aplicações de natureza dinâmica atuais (Ahmad et al., 2015). OpenFlow é a implementação mais aceita e amplamente utilizada no desenvolvimento de arquiteturas de Redes Definidas por Software (Collings; Liu, 2014). Esse amplo uso acontece porque a arquitetura de Redes Definidas por Software , juntamente ao protocolo Openflow, promove o desacoplamento entre plano de dados e plano de controle. (MCKEOWN et al., 2008). Dessa forma, o controle de tráfego é transferido da infraestrutura para o administrador. Como um resultado disso, operadores de rede ganham maiores níveis de controle de rede, automação e otimização com aplicações SDN (Hu; Hao; Bao, 2014).

2.1.1 Visão Conceitual

Na arquitetura de Redes Definidas por Software o plano de controle é separado do plano de dados da rede. A lógica de controle é separada dos dispositivos de encaminhamento, tais como roteadores e switches (Zerrik et al., 2014). Além disso, essa lógica de controle é logicamente centralizada em um controlador SDN. Dessa forma, o plano de dados se torna em um conjunto de simples dispositivos de encaminhamento, os quais são gerenciados pelo controlador (Satasiya; Raviya Rupal D., 2016).

Em SDN, o controlador se comunica e envia instruções, em tempo real, para os elementos de rede. Isso ajuda a corroborar com um dos principais objetivos da arquitetura de Redes Definidas por Software, que é permitir que operadores de rede consigam executar mudanças rápidas de acordo com requisições de áreas de negócio, diretamente no alto nível da arquitetura, a camada de aplicação. As mudanças executadas diretamente no plano de aplicação são redirecionadas, pelo plano de controle, a diferentes blocos de dados dentro do sistema de rede (Jarraya; Madi; Debbabi, 2014). A arquitetura SDN define quatro planos conceituais e interfaces de comunicação, assim como apresentado na Figura 2.1.

Figura 2.1: Arquitetura de Redes Definidas por Software



Fonte: O Autor

2.1.1.1 Plano de Aplicação

SDN permite que aplicações interajam com os dispositivos de rede através da camada de controle. Para isso, as aplicações, no alto nível, fazem uso da visibilidade dos recursos de rede no controlador (Wang et al., 2016).

Em Redes Definidas por Software, o plano de aplicação possui uma visão abstraída e informações dos recursos de todos elementos da topologia da rede. Todas essas informações são providas pelo plano de controle, que se situa em um nível abaixo ma arquitetura (Markowski; Ryba; Puchała, 2016).

2.1.1.2 Plano de Controle

Em Redes Definidas por Software, o Plano de Controle, ao invés de estar presente em cada dispositivo (nodo) da topologia rede, é implementado de maneira logicamente centralizada. Essa funcionalidade do plano de controle é implementada pela entidade representada pelo controlador SDN (Kreutz et al., 2015).

O controlador SDN é responsável por gerenciar toda topologia de rede. Esse gerenciamento é feito através do NOS (*Network Operating System*) (Nam et al., 2019) diretamente de um ponto central da topologia, onde há uma visão global de todos recursos e fluxos da rede (Narantuya et al., 2019).

2.1.1.3 Plano de Dados

A separação do plano de controle e do plano de dados faz com que os dispositivos de encaminhamento, tais como roteadores e pontos de acesso, da rede sejam remotamente controlados via interfaces de controle. Esses dispositivos podem ser configurados de acordo com diversas necessidades da operação de rede (Hu; Hao; Bao, 2014). Essas configurações podem ser feitas via chamada remota diretamente do controlador SDN, fazendo uso de um canal seguro.

Um exemplo de dispositivo de encaminhamento do plano de dados SDN é o switch OpenFlow. Um switch OpenFlow deve ter, entre seus requirimentos mínimos, (i) um canal seguro de comunicação para permitir o recebimento de informações do controlador, (ii) uma tabela de fluxos com ações de cada fluxo, e (iii) um protocolo OpenFlow provendo um mecanismo padrão para a comunicação entre o plano de controle e o plano de dados (Kang et al., 2013).

2.1.1.4 Plano de Gerenciamento

O plano de gerenciamento é responsável pela manutenção e pelo monitoramento do comportamento dos elementos de rede em cada um dos planos conceituais. Além disso, o gerenciamento é focado na configuração desses elementos. Ademais, intervenções humanas podem ser necessárias para executar o gerenciamento de aplicações nesse plano.

Na arquitetura de Redes Definidas por Software, o plano de gerenciamento interage com outros planos conceituais através da utilização de interfaces de comunicação. Além disso, interfaces de gerenciamento também são necessárias, como por exemplo, dis-

ponibilizar uma visão geral integrada dos recursos da topologia de rede com um sistema rodando acima dos mesmos (Wickboldt et al., 2015).

2.1.1.5 Interfaces de Comunicação

A comunicação entre os diferentes planos dentro da arquitetura SDN acontece através de três interfaces: (i) *Southbound*, (ii) *Northbound*, e (iii) de gerenciamento. A interface *Southbound* faz a implementação da comunicação entre o plano de dados e o plano de controle. Através dessa interface, é possível para o plano de controle fazer a configuração de regras de encaminhamento em switches. Essa configuração é baseada em notificações recebidas do plano de dados sobre pacotes de entrada (Farhadi; Du; Nakao, 2014).

Já a interface *Northbound* faz a implementação da comunicação entre o plano de aplicação e o plano de controle. Essa interface faz com que seja possível a programabilidade dentro do controlador SDN. Isso é possível porque a interface expõe as abstrações dos dados das topologias de rede para o plano de aplicação. Atualmente, o protocolo mais utilizado para essa comunicação é o REST (*Representational State Transfer*) (Santos da Silva et al., 2016). Por fim, a interface de gerenciamento é responsável por provisionar a troca de informações entre as diferentes soluções de gerenciamento de rede nos diferentes planos. São exemplos de protocolos para executar essas tarefas o NETCONF[1] e o OF-Config[2] (Wickboldt et al., 2015).

2.1.2 Protocolo OpenFlow

O protocolo OpenFlow é considerado um dos primeiros padrões para redes definidas por software. O mesmo originalmente definiu o protocolo de comunicação em ambientes SDN, permitindo com que o controlador SDN interaja diretamente com o plano de dados de dispositivos de rede (MCKEOWN, 2008).

Esse protocolo, que é um padrão de SDN, foi proposto por Nick Mckeown. A especificação do protocolo foi muito bem aceita pelo meio industrial e pelo meio acadêmico devido a sua facilidade de implantação e possibilidade de inovação dentro de SDN (Panda et al., 2019).

[1] https://tools.ietf.org/html/rfc6241
[2] https://opennetworking.org/technical-communities/areas/specification/of-config/

2.1.2.1 Canal Seguro

Dentro do protocolo OpenFlow, há a possibilidade de criar um canal Seguro para comunicação entre switches e controladores (Sgambelluri et al., 2013). Dentre essas possibilidades se destacam o (i) TLS (Transport Layer Security), (ii) *Secure Shell* e (iii) IPSec (AGBORUBERE; SANCHEZ, 2017).

O TLS tem como mecanismo padrão a autenticação via servidor, resultando na autenticação do controlador OpenFlow. No entanto, o mesmo deixa aberta a possibilidade de *spoofing* dos switches OpenFlow, através da obtenção das suas configurações ou modificação de suas regras. Isso acontece porque as conexões *TCP* sem encriptação são permitidas dentro do TLS (GORANSSON; BLACK, 2014).

Uma outra abordagem para se obter encriptação e autenticação dentro do protocolo OpenFlow é com SSH (*Secure Shell*). Com o protocolo *Secure Shell* é possível gerar chaves e métodos de TUFU (*trust-upon-first-use*) de maneira automática. Ademais, todos os switches são tratados como clientes SSH e o controlador é tratado como seu servidor. O cliente conecta no servidor e se autentica pela sua chave gerada. Nesse processo, ele também autentica a chave do servidor. Então, o túnel de conexão seguro é criado entre a porta de comunicação OpenFlow no lado do servidor (no caso, o controlador) e entre a porta configurada no lado do cliente (no caso, switch) (Vinayakumar; Soman; Poornachandran, 2017).

Uma outra opção de autenticação e criação de um canal seguro com o uso de OpenFlow é via o conjunto de protocolos IPSec. O IPSec, o qual opera na camada de rede, pode ser usado para proteger a transmissão de dados em três modos: (i) *host-to-host*, (ii) *network-to-network*, (iii) *network-to-host* (Hamed; Al-Shaer; Marrero, 2005). Dessa forma, apenas o IPSec, dentre as três opções citadas, fornece proteção em qualquer tráfego dentre aplicações dentro de uma rede de computadores IP, que é rede de comunicação que usa o IP para enviar e receber mensagens entre um ou mais dispositivos de rede.

2.1.2.2 Tabelas de Fluxo

A tabela de fluxo no protocolo OpenFlow consiste de entradas de regras, representadas pelos componentes: (i) *match*, (ii) prioridade, (iii) contadores, (iv) instruções, (v) *timeouts* e (vi) cookie (FOUNDATION, 2012). Os campos de *match* contêm a porta de ingresso no switch e os cabeçalhos de pacotes, além de opcionalmente possuírem metadados referentes a uma tabela associada. Já o campo de prioridade conta com a precedência

de casamento de regras de entrada de fluxo.

Os contadores são os campos que possuem seu valor incrementado quando os pacotes têm regras casadas entre si. Já o campo de instruções é utilizado para fazer modificações no conjunto de ações de determinada regra. O campo referente aos *timeouts* possui o tempo de expiração do fluxo dentro do switch. O campo de cookie, por sua vez, contem valores de dados opacos[3] escolhidos pelo controlador, e deve ser utilizado pelo mesmo para filtrar estatísticas, modificações e deleções dos fluxos contidos na rede.

2.1.2.3 Tipos de Mensagens

As mensagens do protocolo OpenFlow são divididas em duas categorias: (i) assíncronas e (ii) simétricas (FOUNDATION, 2012). Dentre as mensagens assíncronas têm-se (i) *oftp_packet_in*, para pacotes recebidos e mandados ao controlador; (ii) *oftp_flow_removed*, para notificar o controlador se um fluxo atinge seu tempo de expiração ou se é removido da tabela de fluxos; (iii) *oftp_port_status*, para notificar o controlador sobre alterações em portas, além de remoções e adições das mesmas; e (iv) *oftp_error_msg*, para notificar o controlador sobre algum problema no switch OpenFlow.

As mensagens simétricas, por outro lado, são constituídas por (i) *oftp_hello*, que consiste de um cabeçalho OpenFlow mais um conjunto de elementos *hello* de tamanho variável; (ii) *echo request*, a qual consiste de um cabeçalho OpenFlow juntamente a um campo de dados de tamanho arbitrário; (iii) *echo reply*, que consiste também em um cabeçalho OpenFlow, mas com um campo de dados não modificado de uma mensagem de *echo request*; e (iv) *experimenter*, que tem o mesmo cabeçalho das demais mensagens, mas com um campo *experimenter* de 32 bits para identificação (FOUNDATION, 2012).

2.2 Técnicas de Verificação de Regras em Redes de Computadores

Rápidos e constantes avanços na área de verificação de rede têm acontecido nos últimos anos. O gerenciamento consistente de redes de computadores em uma grande escala é um dos maiores desafios dentro da área (Li et al., 2019).

No campo de verificação de redes definidas por software, a verificação do plano de dados tem atraído uma maior atenção, uma vez que esse plano é mais natural para ser modelado com uma semântica bem definida, em comparação com o plano de controle

[3]Tipos de dados os quais suas estruturas de dados não são definidas por uma interface.

(Fang; Lu, 2020). Já em redes de computadores convencionais, um *snapshot* (registro instantâneo de um fluxo) de suas configurações e topologias pode ser coletado para uma verificação *offline* (XIE et al., 2005).

2.2.1 Verificação Formal

Verificação formal é uma técnica de utilização de prova de corretude de implementação com respeito a propriedades e requirimentos de regras, onde esses devem ser satisfeitos com o uso de métodos ou descrições formais (Ivutin; Voloshko, 2020). Essas descrições formais são especificações expressadas em linguagens as quais têm suas semânticas formalmente definidas, assim como as suas sintaxes e os seus vocabulários (Xiang; Jianlin, 2011).

Geralmente, inclui técnicas como verificação de modelos (HSIEH; LEVITAN, 1998), simulação simbólica (RADOJICIC; PURUSOTHAMAN; GRIMM, 2015) e reduções *SAT* (PATHAK; TIWARI; CHAUDHARI, 2011). Como desvantagem, muitas vezes a interferência manual é necessária para adicionar axiomas[4] que o sistema não pode considerar automaticamente (MAJUMDAR; TETALI; WANG, 2014). Já no contexto de redes de computadores, em geral, a verificação formal é utilizada para checar a corretude de configurações de rede fixa, como o comportamento de encaminhamento de um determinado pacote (Shaoying Liu, 2003).

2.2.2 Verificação Estática

Verificação estática é um conjunto de técnicas que utiliza análise estática baseada em transformações matemáticas (Ge et al., 2011). No contexto de redes de computadores, esse conjunto pode ser usado para identificar e testar regras formadas por fluxos de rede que estão circulando, na mesma, em um determinado instante.

Em Redes Definidas por Software, para capturar as informações dos dados, que estão trafegando na rede em um intervalo de tempo específico, faz-se a utilização de *snapshots* (Yang; Yang; Tu, 2019). A partir da captura dos *snapshots*, é possível fazer a verificação das regras contidas nos mesmos. As regras se apresentam na forma lógica, e podem ser validadas com o uso de verificação formal estática.

[4]Expressões universalmente válidas e verdadeiras.

2.3 Plataformas de Gerenciamento de Redes

Plataformas de gerenciamento de redes de computadores vêm sendo difundidas dentro do meio acadêmico e dentro da indústria. Essa popularidade tem ocorrido devido a grande demanda por tecnologias que supram a alta variedade de elementos existentes nesse contexto (Caiazza et al., 2020).

No entanto, problemas como baixa eficiência e flexibilidade limitada ainda são encontrados no desenvolvimento de ferramentas nessa área (Wang et al., 2016). Dentre as áreas onde as plataformas para gerenciamento de rede atuam, destacam-se, no meio acadêmico, as desenvolvidas para o monitoramento de redes. Já no meio industrial, destacam-se as para monitoramento de infraestrutura de sistemas e de fluxos dentro de redes de computadores.

2.3.1 Plataformas de Monitoramento de Redes

No contexto de monitoramento de redes, mais especificamente na performance de servidores, destaca-se o Framework of Network Management System (Ismail; Syarmila, 2009). Esse trabalho tem como objetivo, além do monitoramento dos dispositivos, a coleta de medidas relacionadas à performance dos mesmos, para então, os classificar de acordo com a quantidade de dados trafegadas neles.

Além de monitorar a performance dos servidores, o framework faz o monitoramento de velocidade de *upload* e *download* dentro da rede. Ademais, o mesmo auxilia os operadores de rede a preparar e propor novas ações sistemáticas dentro da topologia. Assim como o Placidus, o framework proposto por proposto por Ismail e Syamila é expansível e escalável, de maneira a permitir que outros desenvolvedores contribuam para o seu crescimento.

Já no contexto de plataformas de gerenciamento de redes, um trabalho com bastante relevância é a ferramenta *open source* Zenoss Core[5]. Essa ferramenta é utilizada, principalmente, para monitorar redes de computadores heterogêneas de maneira centralizada (Ljubojević; Bajić; Mijić, 2018).

Assim como a plataforma apresentada nessa monografia, o Zenoss permite, de maneira eficiente, o monitoramento e a visualização de dispositivos da rede (Kazaz et al., 2012). Isso permite uma manipulação e representação apropriada dos dados trafegados e

[5]https://www.zenoss.com

a verificação de requisitos da rede como um todo. Além disso, fornece aos operadores de rede uma visão sistemática da topologia como um todo, auxiliando os mesmos na tomadas de decisão em alto nível (Ismail; Syarmila, 2009).

2.3.2 Plataformas de Gerenciamento na Indústria

Plataformas de Gerenciamento de redes de computadores também foram desenvolvidas para a indústria. Uma delas é o Nagios[6], uma plataforma de gerenciamento e monitoramento de infraestrutura de sistemas de TI. O monitoramento feito pelo Nagios acontece antes do estágio de execução dos sistemas dentro das infraestruturas configuradas. de maneira crítica por decorrência de configurações erradas ou *malwares* nos seus dados trafegados.

Assim como no Placidus, com o Nagios, é possível monitorar diversos nodos dentro de uma topologia, além de fazer a constante verificação do estado atual da rede monitorada (Renita; Elizabeth, 2017). Diferentemente do Placidus, o Nagios, em sua versão atual, não possui um sistema de usuários dentro da plataforma. Para fazer uso do Nagios, cada usuário deve efetuar o *download* diretamente do website oficial da plataforma, e instalá-lo de maneira local nos servidores onde o monitoramento deve ser realizado de maneira ativa ou passiva.

Outro trabalho com bastante relevância na indústria é o Zabbix[7], que é uma plataforma para monitoramento de fluxos dentro de redes de computadores. Assim como a plataforma apresentada nesse trabalho, o Zabbix está disponível na nuvem, podendo ser acessado de qualquer lugar sem a necessidade de instalação por parte dos usuários.

Diferentemente do Placidus, o Zabbix não faz verificação de propriedades nos fluxos de rede, tais como conflitos e redundâncias. No entanto, o Zabbix permite que operadores de rede aumentem a eficiência de seus testes, uma vez que disponibiliza um controle de acesso a todos os dispositivos interconectados dentro de um ambiente de rede (Petruti et al., 2018).

[6]https://www.nagios.com
[7]https://www.zabbix.com

3 PLATAFORMA DE GERENCIAMENTO COM VERIFICAÇÃO FORMAL PARA REDES DEFINIDAS POR SOFTWARE

Neste capítulo feita a apresentação de uma plataforma desenvolvida para atuar no gerenciamento e na verificação de regras em arquiteturas de Redes Definidas por Software, o Placidus. Este capítulo está dividido da seguinte forma: A Seção 3.1 apresenta a análise de requisitos feita no início da implementação da plataforma. Já a Seção 3.2 descreve as estratégias de verificação utilizadas no núcleo do Placidus. A Seção 3.3, por fim, apresenta a arquitetura proposta para a plataforma desenvolvida nesse trabalho.

3.1 Estudo de Requisitos

Durante a execução desse trabalho, foi identificada a necessidade de se desenvolver uma plataforma que abrangesse desde o gerenciamento de redes até a verificação de regras contidas dentro das topologias. Essa necessidade se deve principalmente ao fato de se possuir uma plataforma que facilitasse a manipulação de fluxos em topologias complexas por operadores de rede.

Com uma plataforma de gerenciamento e verificação, operadores de rede podem criar topologias de teste, e verificar regras e fluxos dentro das redes definidas antes de montar um ambiente real. Dessa forma, uma redução considerável de tempo e recursos computacionais é feita, uma vez que erros podem ser corrigidos antes da montagem das topologias em infraestruturas de produção.

3.1.1 Framework modular de verificação de regras

Ao longo do desenvolvimento do projeto apresentado nesse trabalho, foi identificado a importância de se agrupar mais de um tipo de verificação de regras no mesmo sistema. Além disso, notou-se a relevância de possuir um sistema modular, onde diferentes tipos de verificações de regras fossem adicionadas de maneira simples e funcional.

Para isso, ao invés de haver inúmeros sistemas e *scripts* para executar a checagem de diferentes propriedades (e.g.: conflitos e redundâncias entre regras), foi escolhido, para esse trabalho, uma estrutura no formato de *framework* para a criação de diferentes módulos de verificação. Dessa maneira, cada módulo é responsável por um determinado

tipo de execução, o que diminui o acoplamento e facilita o desenvolvimento de novas funcionalidades dentro do sistema.

3.1.2 Plataforma online de gerenciamento de redes

Dentro da plataforma de gerenciamento de topologias de rede, foi notada a necessidade de um sistema de fácil acesso pelos operadores. Para isso, foi identificado que era preciso que o sistema estivesse disponível aos usuários de qualquer lugar, de maneira online.

Além disso, os serviços implementados devem ser disponibilizados com segurança aos usuários. Para que o sistema seja seguro e para que usuários não acessem informações sigilosas de outros operadores, foi identificada a necessidade de se implementar uma autenticação via login e senha dentro da plataforma. Ademais, com autenticação, é possível que cada usuário tenha acesso aos seus ambientes de testes, podendo assim, compartilhá-los com outros usuários da plataforma. Isso é possível porque cada usuário teria o seu ambiente associado com a sua chave de login utilizada no registro.

3.2 Estratégias de Verificação Formal do Placidus

Estratégias de verificação formal de regras e fluxos de rede compõem o núcleo da plataforma desenvolvida nesse trabalho, denominada Placidus. A plataforma conta com módulos de verificação para diferentes propriedades dentro de fluxos de rede.

Dentre as propriedades verificadas pelo núcleo da plataforma, têm-se: (i) conflitos e (ii) redundâncias entre regras, e (iii) alcançabilidade de pacotes dentro de uma topologia de rede. As duas primeiras propriedades podem ser verificadas formalmente para tabelas de fluxos SDN e para firewalls[1]. Para verificar se ocorre conflito ou redundância entre regras, analisa-se o *match* corpo das mesmas, identificando a igualdade entre duas ou mais regras, decidindo-se pela ação da regra se é um conflito ou redundância em um mesmo dispositivo (Kim; Kang, 2020). No caso de regras com corpo igual, mas ações diferentes, é considerado um conflito. Para regras com corpos e ações iguais, uma redundância é considerada na análise.

Já a propriedade de alcançabilidade pode ser verificada no plano de dados dispa-

[1]Equipamento que forma uma barreira em redes de computadores para controlar o fluxo de dados entre um link da internet e a rede interna.

rando um pacote de um ponto A até um ponto B dentro da topologia de rede. A alcançabilidade de um pacote significa que um pacote enviado de uma origem A consegue alcançar um destino B através de um certo caminho (Zhang et al., 2020).

3.2.1 Módulo de verificação formal de conflitos e redundâncias

Um dos módulos do núcleo de verificação formal da Plataforma Placidus é o responsável por duas verificações de fluxos dentro das topologias de rede. Esses fluxos dentro da rede são modelados internamente como regras lógicas. Ademais, esse módulo é sub-dividido de maneira a abranger a verificação de duas propriedades de regras: conflitos e redundâncias. Além disso, esses sub-módulos fazem a verificação de fluxos tanto para tabelas de fluxos presentes no controlador SDN, quanto para regras implementadas nos firewalls presentes nas topologias implementadas.

Essa verificação de propriedades é baseada em sucessivas comparações, duas a duas, dos dados de reconhecimento de pacotes de cada regra. Dentro do módulo, cada regra é modelada como um predicado, o qual contêm inúmeros predicados atômicos. Esses predicados atômicos simbolizam as informações específicas de uma determinada regra (Zhang et al., 2020). Nas comparações feitas durante a execução do algoritmo de verificação, as regras são agrupadas duas a duas, a cada iteração, tendo seus índices representados por *"cont1"* e *"cont2"*, como ilustrado na Figura 3.1. Essas comparações entre os *matches* das regras são feitas de maneira exaustiva, utilizando todos predicados atômicos contidos nos dados de reconhecimento de pacotes de cada regra verificada.

Por meio de inúmeras iterações, variando linearmente com o número de regras contidas na rede, são feitas verificações de igualdade de informações dos predicados atômicos entre as duplas de regras que estão sendo comparadas. Caso todos predicados atômicos da dupla sejam idênticos, é detectado conflito ou redundância entre regras. Um conflito é detectado entre duas regras na iteração quando os seus *matches* comparados são idênticos, mas suas ações implicadas são diferentes. Por sua vez, uma redundância ocorre quando os *matches* comparados e as regras implicadas por cada predicado são iguais.

Caso uma regra englobe alguma outra durante uma iteração, os predicados atômicos que são *wildcards* (expressões curingas) recebem os valores dos predicados atômicos que não são. Conforme pode ser visto no quadro a seguir, antes das comparações, o quarto predicado atômico de Regra 2 (X) é uma expressão curinga. Entretanto, depois das comparações, o mesmo recebe o valor que está no quarto predicado atômico de Regra

1 (*2*). Isso acontece porque no núcleo de verificação de regras do Placidus, como são comparadas duas regras por vez a cada iteração, se uma das regras comparadas possui um predicado atômico curinga, o valor atribuído para esse predicado atômico é o valor contido no predicado atômico da outra regra comparado. Nota-se, nesse exemplo, que após a atribuição do valor 2 ao predicado atômico que possuía uma *wildcard*, que Regra 1 e Regra 2 são consideradas conflitantes. Isso acontece porque, depois da atribuição, ambas as regras possuem os mesmos dados reconhecimento de pacote (valores antes da implicação), porém valores de ação diferente (*ALLOW* e *DROP*).

Antes das comparações :

Regra 0: $00{:}00{:}00{:}00{:}00{:}0e \wedge 00{:}00{:}00{:}00{:}00{:}8a \wedge 0x0x806 \wedge 8 \implies ALLOW$

Regra 1: $00{:}00{:}00{:}00{:}00{:}78 \wedge 00{:}00{:}00{:}00{:}00{:}8c \wedge 0x0x806 \wedge 2 \implies DROP$

Regra 2: $00{:}00{:}00{:}00{:}00{:}78 \wedge 00{:}00{:}00{:}00{:}00{:}8c \wedge 0x0x806 \wedge X \implies ALLOW$

Depois das comparações:

Regra 0: $00{:}00{:}00{:}00{:}00{:}0e \wedge 00{:}00{:}00{:}00{:}00{:}8a \wedge 0x0x806 \wedge 2 \implies ALLOW$

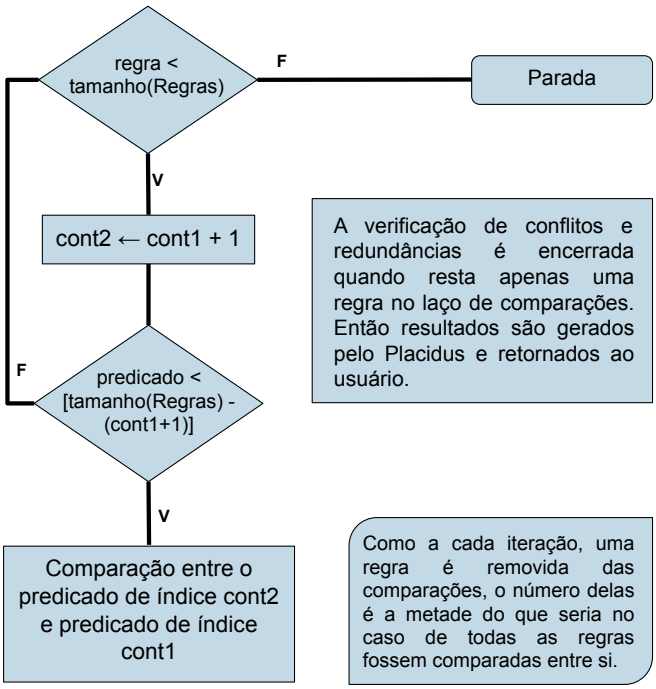
Regra 1: $00{:}00{:}00{:}00{:}00{:}78 \wedge 00{:}00{:}00{:}00{:}00{:}8c \wedge 0x0x806 \wedge 2 \implies DROP$

Regra 2: $00{:}00{:}00{:}00{:}00{:}78 \wedge 00{:}00{:}00{:}00{:}00{:}8c \wedge 0x0x806 \wedge 2 \implies ALLOW$

Para diminuir a quantidade de comparações executadas pelo algoritmo de verificação, a cada iteração, a primeira regra da dupla agrupada é removida. Essa remoção de uma regra a cada iteração acarreta em uma redução de 50% no número de comparações em relação à quantidade de comparações utilizadas por algoritmos que comparam todas as regras entre si, como é o caso do trabalho *Formal Verification of Firewall Policies* (LIU, 2005), que verifica regras presentes em firewalls. A Figura 3.1 apresenta um fluxograma que ilustra como as comparações entre regras acontecem nesse núcleo de verificação. Ademais, o Algoritmo 1 apresenta um pseudo-código da verificação executada nesse módulo.

Após a verificação completa dos fluxos da rede, três arquivos são gerados: (i) com regras conflitantes, (ii) com regras redundantes e (iii) com todas as regras contidas na rede. Cada linha desse arquivo é representada na forma de um conjunto de predicados atômicos que representam as informações de uma regra, unidos pelo operador lógico *AND* ($\wedge$). Além disso, esse conjunto de regras, em cada linha, implica em uma ação. Um exemplo do formato utilizado em cada linha do arquivo de saída é apresentado no bloco a seguir, onde é apresentada uma regra presente na tabela de encaminhamento de um switch.

Figura 3.1: Fluxograma de comparação entre regras do módulo de conflitos e redundâncias no Placidus



Fonte: O Autor

$$00{:}00{:}00{:}00{:}00{:}8a \wedge 00{:}00{:}00{:}00{:}00{:}7d \wedge 0x0x806 \wedge 11 \implies output = 9$$

3.2.2 Módulo de verificação formal de alcançabilidade

O segundo módulo do núcleo de verificação do Placidus é o de verificação formal de *reachability* (alcançabilidade). Essa propriedade é verificada, no contexto desse trabalho, a partir do momento em que um pacote sai de uma origem e consegue chegar no destino especificado em seus dados de reconhecimento (*match*) (XIE et al., 2005).

A primeira etapa do processo de verificação dessa propriedade no módulo é a conversão das células da tabela de fluxos de rede. Todas as células, as quais estão no formato de strings hexadecimais, são convertidas em vetores de *bits*. Após ter uma tabela com vetores de *bits* com os dados da rede, a mesma é utilizada para a montagem de um grafo de fluxos que a representa. Cada nodo desse grafo possui as seguintes informações: (i) *match*, (ii) destino, (iii) ação ou saída, (iv) switch e (v) um *bit* de visitado, o qual é utilizado em uma busca em largura no grafo da topologia. O grafo montado durante esse processo representa como os fluxos da topologia se relacionam entre si. Nodos conecta-

Algorithm 1 Algoritmo de Verificação de Conflitos e Redundâncias

```
1: para l faça tamanho(listaAções)-1
2:     j ← i + 1
3:     para regra faça tamanho(listaAções)-(i+1)
4:         se PredicadoAtômico[i] == 'x' então
5:             PredicadoAtômico[i] ← PredicadoAtômico[j]
6:         fim se
7:         se Match[i] == Match[j] então
8:             se Ação[i] != Ação[j] então
9:                 Conflito detectado
10:            senão
11:                Redundância detectada
12:            fim se
13:        fim se
14:    fim para
15: fim para
```

dos, para o algoritmo de verificação quando executado, representam dados de dispositivos conectados e alcançáveis entre si.

Tendo o grafo completo da topologia representado em vetores de *bits*, o módulo inicia o processo de verificação da propriedade de alcançabilidade para um determinado pacote de entrada. Cada pacote contem duas informações: (i) *match* e (ii) destino. O algoritmo de verificação implementado para esse processo consiste em fazer uma busca em largura desse pacote no grafo montado. O *match* é utilizado para achar o nodo em que inicia a busca. Já o destino é utilizado como informação de parada no algoritmo. Nesse processo, a propriedade de *reachability* está correta, ao final da busca, se: (i) O destino do pacote for igual ao da última regra, (ii) o destino da regra pertencer ao switch que contém a mesma, e (iii) o *bit* de visitado do último nodo estiver ligado.

A Figura 3.2 apresenta um fluxograma que ilustra como ocorre a verificação formal de alcançabilidade dentro do núcleo do Placidus. No primeiro passo, tem-se a conversão dos valores em hexadecimal, providos pelo controlador SDN, em vetores de bits. Já no segundo passo, é criado o grafo de fluxos que representam a topologia. Já na parte de baixo da imagem, é apresentado como os testes de verificação de alcançabilidade são executados. Primeiramente, um pacote de entrada contendo as informações de reconhecimento de pacote é inserido na rede. A partir dessas informações, uma busca em largura desse pacote é feita no grafo de fluxos da topologia.

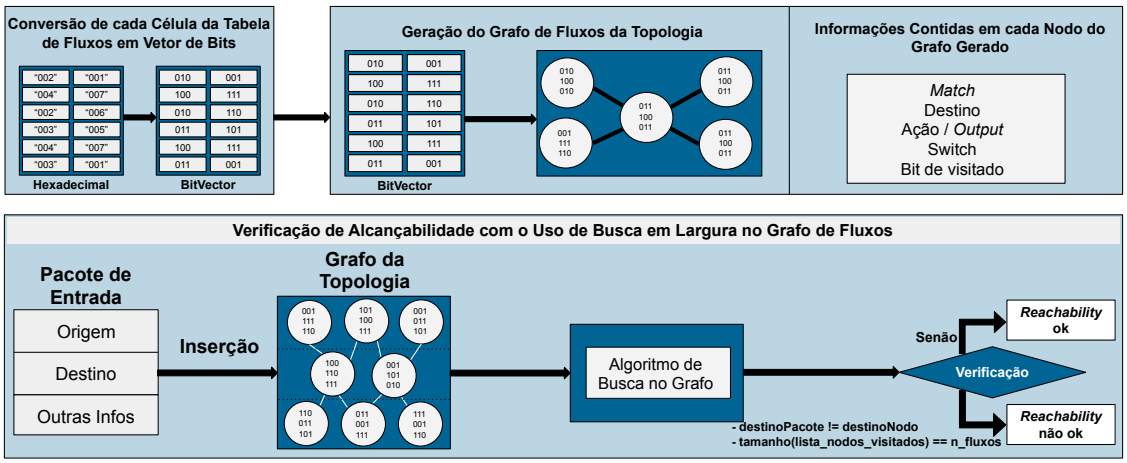
O Algoritmo 2 apresenta um pseudo-código dessa busca executada pelo pacote nessa verificação. Nesse algoritmo, é executada uma recursão de uma busca em largura para o grafo de fluxos. Primeiramente, o novo inicial de busca recebe os dados do pacote

inserido na topologia Na sequência, para cada fluxo dentro da rede, é feita a obtenção dos dados do nodo visitado, onde os dados do nodo visitado são momentaneamente armazenados em memória para continuar a busca. Depois, o bit de visitado do nodo é atualizado para o valor lógico verdadeiro, e a lista de nodos visitados é atualizada com a adição desse novo nodo. Por fim, o destino do nodo é verificado para saber se o mesmo já conseguiu alcançar o seu destino, e o mesmo é atualizado para próximo nodo a ser visitado. Então, uma nova busca é refeita com os dados atualizados. Caso, ao final da execução, a lista de nodos visitados possui o mesmo tamanho do número de nodos da topologia, mas o destino no pacote não foi encontrado, a alcançabilidade é dada como falsa para o destino que estava no pacote que iniciou a busca, ou verdadeira no caso contrário.

Algorithm 2 Algoritmo de Busca de Pacote na Topologia

```
1:  função BUSCA_LARGURA_GRAFO(pacote, topologia, lista_bit_visitado)
2:      nodo = obtem_dados (pacote)
3:      para fluxos faça fluxo
4:          obtem_dados_reconhecimento_nodo (nodo)
5:          atualiza_bit_visitado_nodo (nodo.bit_visitado)
6:          atualiza_lista_nodos_visitados (lista_bit_visitado, nodo)
7:          verifica_destino_nodo (nodo, pacote)
8:          atualiza_novo_nodo (nodo)
9:          devolve BUSCA_LARGURA_GRAFO (nodo, topologia, lista_bit_visitado)
10:     fim para
11: fim função
```

Figura 3.2: Fluxograma de verificação de alcançabilidade no Placidus



Fonte: O Autor

Essa conversão de todas as células em vetores de bits foi feita nesse módulo para diminuir a quantidade de memória e tempo utilizada na execução do algoritmo de verificação. Essa tentativa de diminuir a quantidade de tempo e memória usada foi tomada

porque no módulo de verificação de regras, o qual foi primeiramente desenvolvido nesse trabalho, ocorreu um *overhead* maior na verificação por conta da modelagem das regras na forma de strings de predicados. Após fazer a comparação dos resultados dos dois módulos, os quais são apresentados no capítulo seguinte, decidiu-se ter como trabalho futuro a uniformização das estruturas de dados. Essa uniformização será feita para que os dois módulos possam fazer uso dos vetores de bits em suas comparações durante a execução de seus algoritmos.

3.2.3 Expansão do Núcleo de Verificação

O Placidus possui atualmente em seu núcleo de verificação, verificadores de regras (conflitos e redundâncias) para tabelas de encaminhamento de switches e para regras de firewalls. Além disso, o núcleo de verificação do Placidus contém um módulo para verificação de alcançabilidade para topologias de rede virtual. O núcleo da verificação do Placidus é modular, e os módulos de verificação presentes nesse não são dependentes entre si. Essa modularização no núcleo permite uma expansão de verificadores dentro da plataforma de forma mais organizada e efetiva.

Como o foco nesse trabalho foi o de construir uma plataforma de gerenciamento de SDN, uma quantidade maior de verificações de propriedades será contemplada no Placidus em trabalhos futuros. Todavia, outras propriedades já foram estudadas para serem verificadas pelo núcleo de verificação do Placidus, assim como estarem disponíveis no dashboard da plataforma.

Dentre as verificações que estão sendo estudadas para serem implementadas dentro do Placidus, tem-se, por exemplo, a de *slice isolation*, que é a verificação de um pacote pertencer a um determinado pedaço da topologia de rede (KAZEMIAN; VARGHESE; MCKEOWN, 2012). Ademais, outra verificação que está na fila para ser implementada é a de detecção de *black holes*, os quais são sumidouros de pacotes trafegados dentro da rede (Kompella et al., 2007). Além disso, deseja-se ter dentro do núcleo de verificação a detecção de *loops* de pacotes dentro da rede, os quais acontecem quando um pacote retorna a uma porta que já havia sido visitada (KAZEMIAN; VARGHESE; MCKEOWN, 2012).

Todos esses novos verificadores seguirão o mesmo padrão de estruturas de dados do verificador de alcançabilidade, fazendo uso da representação de vetores de bits. Além disso, a representação das topologias de rede para os três novos verificadores será

na forma de grafo, permitindo o reaproveitamento dos serviços de busca em largura já implementados para módulo de verificação de alcançabilidade. Por fim, os novos verificadores estarão também presentes no dashboard da Plataforma Placidus, nos submenus de verificação formal da mesma. Para isso, seguirão a mesma estrutura de arquitetura proposta para os outros verificadores já implementados.

3.3 Arquitetura do Placidus

Um dos principais pré-requisitos antes do desenvolvimento do Placidus era o de o mesmo possuir uma arquitetura modular e facilmente escalável. Para se obter escalabilidade no núcleo de verificação formal, optou-se pela separação dos verificadores em diferentes módulos, os quais não possuem dependência entre si, contribuindo para um baixo acoplamento entre esses blocos verificadores.

Além do núcleo de verificação formal, estratégias de organização de módulos foram utilizadas nos níveis mais altos e de integração dentro da Plataforma Placidus. Para atingir esse objetivo, a plataforma foi separada em diversos componentes independentes, os quais se comunicam via API com os seus respectivos serviços internos e externos ao Placidus.

A Figura 3.6 mostra uma visão geral da arquitetura do Placidus. Ela contém componentes externos, os quais possuem funcionalidades oferecidas a partir de ferramentas e plataformas de terceiros. Além disso, possui componentes denominados nesse trabalho como internos, os quais foram desenvolvidos unicamente para o escopo da Plataforma Placidus. Os principais componentes são (i) o de coleta de dados (utilizado por todo o núcleo de verificação), (ii) o de integrações REST APIs (por permitir a conexão entre o dashboard e os serviços do *back end* da plataforma), e o (iii) Dashboard UI (por permitir que os usuários acessem a todas as funcionalidades da plataforma de maneira simples). A seguir, os componentes da arquitetura serão apresentados em detalhes.

3.3.1 Componentes Externos Integrados

A arquitetura proposta para o Placidus é baseada na modularização do sistema inteiro em componentes independentes. Essa separação em módulos é feita tanto para o *front end* quanto para o *back end* da plataforma. Os serviços externos ao Placidus auxiliam

no funcionamento da plataforma e atuam de maneira a deixar a comunicação entre os diferentes módulos mais fluída e robusta. O Placidus possui três sistemas externos: (i) o Google Firebase[2], (ii) o controlador SDN Floodlight[3], e (iii) o emulador de rede Mininet[4].

O Google Firebase é uma plataforma para criação e expansão de aplicações móveis e web. Dentro do Placidus, o Google Firebase tem duas utilidades. A primeira delas é a de controle de usuários dentro da Plataforma Placidus. Esse controle é feito pela autenticação provida pelo Firebase, a qual é utilizada no registro de novos usuários e no acesso dos mesmos à plataforma. A segunda utilidade do Firebase explorada nesse trabalho é a de banco de dados em tempo real. Esse segundo uso do Firebase é explorado no dashboard do Placidus. Esse banco de dados é do tipo NoSQL e é hospedado na nuvem, permitindo seu acesso remoto de qualquer lugar, assim como a plataforma apresentada nesse trabalho. Dentro da plataforma, os seguintes dados são armazenados e acessíveis: (i) informações dos usuários, (ii) topologias de rede implantadas e (iii) dados das verificações formais efetuadas. O segundo componente externo integrado ao Placidus é o controlador SDN OpenFlow Floodlight.

O Floodlight é um controlador para ambientes de Redes Definidas por Software desenvolvido por uma comunidade aberta de desenvolvedores. Esse controlador SDN é responsável por manter todas as regras da topologia e por prover, para as camadas subjacentes, as instruções necessárias de como o tráfego de rede deve ser manipulado. O Placidus faz uso dessas regras obtidas pelo Floodlight através do componente de coleta de dados.

Por fim, o terceiro componente externo integrado ao Placidus é o emulador de redes Mininet. Esse é utilizado para fazer uma rápida prototipação de topologias SDN. O Mininet permite a emulação completa de redes com a criação de *hosts*, switches e ligações entre eles dentro de uma única máquina. A integração do emulador e do Placidus é feita através do componente interno de REST API contido na plataforma. Com essa integração, os usuários conseguem fazer a emulação completa de topologias de rede através do dashboard do Placidus.

[2]https://firebase.google.com
[3]https://floodlight.atlassian.net
[4]http://mininet.org

3.3.2 Componentes Internos

Além do bloco de componentes externos, o Placidus também contém um bloco de componentes internos. Esse bloco é responsável por tarefas que tangem desde o lado do cliente da plataforma, até o lado do servidor da mesma.

Mais especificamente, os componentes internos são responsáveis por: (i) apresentar uma interface gráfica aos usuários, (ii) pela comunicação entre o *front end* da plataforma e o *back end* da mesma via REST API, (iii) pela coleta e tratamento de dados do controlador SDN, e (iv) pelo núcleo de serviços de verificação formal.

3.3.2.1 Interface Gráfica

O componente no mais alto nível de abstração dentro do Placidus é o de interface gráfica. Esse componente é responsável pelo dashboard do Placidus, e conta com uma interface de fácil acesso às funcionalidades oferecidas pelos serviços disponíveis dentro da plataforma apresentada nesse trabalho. O dashboard é dividido em duas áreas distintas: (i) área deslogada, e (ii) área logada.

Na área deslogada o usuário pode fazer o *login* na plataforma, caso já tenham criado seu acesso previamente. Caso o usuário ainda não possua uma conta no Placidus, o mesmo pode criar uma na página de registro. Nesse registro, o usuário precisa apenas informar um nome de usuário, um e-mail e uma senha de acesso. Após informar os dados necessários e os submeter pela tela de registro, esses são encaminhados à base de dados provida pelo componente externo Google Firebase. Além disso, um acesso à plataforma é criado junto à autenticação utilizada nesse trabalho.

Já na área logada, o usuário tem acesso a diferentes funcionalidades da Plataforma Placidus. Dentre as funcionalidades presentes, encontram-se (i) o controle de conta, (ii) o controlador SDN, (iii) as topologias de rede, (iv) o gerenciador de firewall, e o (v) o menu de verificação formal.

3.3.2.2 Integração REST API

Em um nível abaixo na arquitetura do Placidus encontra-se o componente de integração de serviços via REST API. Esse componente é responsável pela comunicação entre a interface do usuário e os serviços disponíveis no núcleo de verificação. Ademais, esse componente faz a integração do nível de abstração mais alto com os componentes

externos integrados ao Placidus, deixando as configurações dos mesmos transparentes aos operadores de rede que utilizam a plataforma.

Esse componente é sub-dividido entre serviços e controladores. A camada de serviços é responsável pela lógica das integrações com os componentes internos e externos. Dentro da camada de serviços encontram-se os métodos responsáveis pela conexão com o componente externo Mininet, o qual é utilizado para a emulação de redes virtuais no Placidus. Dentre as responsabilidades desses métodos, destacam-se a criação de redes virtuais e o término das mesmas. Para que fosse possível a criação de redes virtuais de maneira dinâmica, o serviço responsável faz a utilização de uma suspensão momentânea de um segundo dentro dos laços de repetições para fazer as criações das ligações entre hosts e switches implantados nas topologias, garantindo o funcionamento esperado para as camadas de maior abstração do sistema. Já para desligar as redes virtuais criadas, o serviço garante o fechamento da rede com o uso das funções internas de parada do Mininet. Ademais, o serviço se certifica do término de todos os processos ligados à topologia criada.

Além dos métodos de integração com o emulador Mininet, a camada de serviços também conta com integrações com os módulos presentes no núcleo de verificação da plataforma. Os serviços dessa parte da camada contam com as verificações formais de regras e alcançabilidade. Para isso, os mesmos passam os parâmetros providos pela interface gráfica ao núcleo de verificação. Após as verificações serem completadas, os serviços tratam os respectivos retornos, e os enviam para o nível de abstração mais acima na arquitetura.

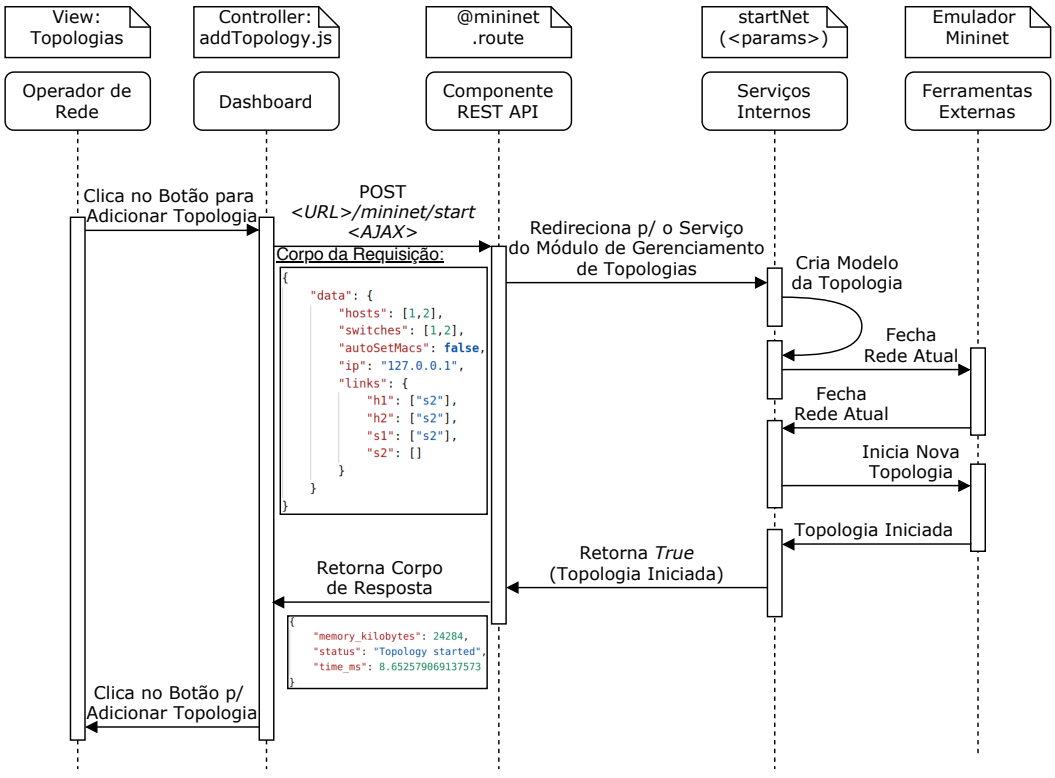
Todo o acesso à camada de serviços feito pela interface gráfica de plataforma é gerido pela camada de controle. É dentro dessa camada que encontram-se as rotas de acesso (*endpoints*) para chegar nos serviços. A camada de controle não possui função lógica, uma vez que a mesma apenas gerencia as requisições aos e respostas dos serviços. A Figura 3.3 mostra um diagrama de sequência com um exemplo do uso do componente de integração via REST API dentro da Plataforma Placidus. Nesse exemplo, é apresentado o fluxo para criação de uma topologia de rede virtual dentro do gerenciador de topologias da plataforma.

No diagrama, é possível verificar o fluxo desde o operador de rede utilizando o dashboard da plataforma até o nível mais baixo, que é o de conexão com o emulador externo de rede Mininet. O componente REST API recebe os dados da requisição do controlador do dashboard no formato JSON (*JavaScript Object Notation*) na rota definida

para criação de topologia. Após isso, o componente processa os dados recebidos no corpo da requisição e os envia para o serviço interno, o qual se comunica diretamente com o emulador Mininet, de maneira a enviar os comandos necessários para a criação de uma topologia de rede virtual.

Após a última camada ter sua execução finalizada, o serviço que a acessava retorna a confirmação para a rota, e a mesma envia uma resposta, também no formato JSON, para o controlador do dashboard. Por fim, o controlador do dashboard fornece um retorno para o operador de rede que iniciou o processo. Esse padrão de fluxo apresentado é o mesmo utilizado nas outras funcionalidades da plataforma que se comunicam com o *back end* via REST API.

Figura 3.3: Exemplo de uso do componente de integração via REST API dentro do Placidus



Fonte: O Autor

3.3.2.3 Coleta e Tratamento de Dados

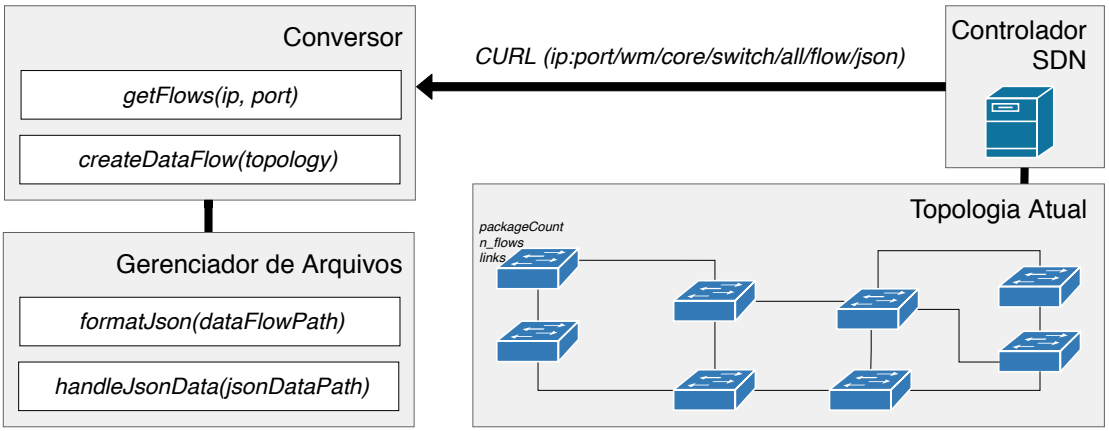
O componente de coleta de dados do Placidus é responsável por executar o acesso aos dados providos pelo controlador SDN utilizado, o qual foi apresentado anteriormente nesse trabalho. Além de acessar os dados, esse componente faz a conversão dos mes-

mos para os formatos usados dentro dos serviços do núcleo de verificação da Plataforma Placidus.

Para coletar os dados, o componente de integração REST API é utilizado. Isso é possível porque o controlador Floodlight tem uma REST API para disponibilização de suas funcionalidades. Dentre as informações obtidas pelo componente de coleta de dados estão: (i) regras de encaminhamento de pacotes dos switches, e (ii) obtenção das regras do firewall. Além da obtenção de dados, o componente também faz o tratamento dos mesmos. Esse tratamento é feito na forma de conversão dos dados em formatos que permitem uma mais fácil manipulação por parte dos serviços que o utilizam. Ademais, o módulo é responsável pelo tratamento dos dados retornados pelos serviços de verificação formal em um formato de lógica de predicados, que é mais facilmente lido pelos usuários da plataforma.

A Figura 3.4 apresenta a coleta e o tratamento de dados dentro da Plataforma Placidus, apresentando os métodos presentes em cada etapa. Primeiramente, o módulo de coleta de dados obtém do controlador SDN um arquivo JSON com a descrição dos fluxos contidos na topologia de rede. O arquivo JSON contido na Figura 3.5 ilustra o formato do arquivo obtido do controlador SDN utilizado na Plataforma Placidus.

Figura 3.4: Coleta e Tratamento de Dados dentro do Placidus



Fonte: O Autor

Após a obtenção do arquivo com os fluxos, os mesmos são extraídos, e convertidos internamente para objetos em memória de maneira em que os serviços internos do Placidus possam os manipular. Depois dos serviços de verificação terem sido utilizados, os arquivos de resultados manipulados e gerados pelo componente são exportados, em formato CSV (*Comma-Separated Values*), para que possam ser vistos pelos operadores de rede que utilizam a plataforma. O formato utilizado nos arquivos exportados pode ser

Figura 3.5: Formato do arquivo de fluxos recebido do controlador SDN

Fonte: O Autor

visto no bloco a seguir.

$$[Switch : 00{:}00{:}00{:}00{:}00{:}00{:}00{:}01]$$
$$(0 \wedge 2) \implies output = 2$$

3.3.2.4 Núcleo de Serviços de Verificação Formal

Por fim, o último componente interno do Placidus é o de serviços do núcleo de verificação formal. Esse componente conta com dois sub-módulos: (i) Verificador de Conflitos e Redundâncias e (ii) Verificador de Alcançabilidade. Esses sub-módulos atualmente possuem diferentes estruturas e métodos para tratamento de dados.

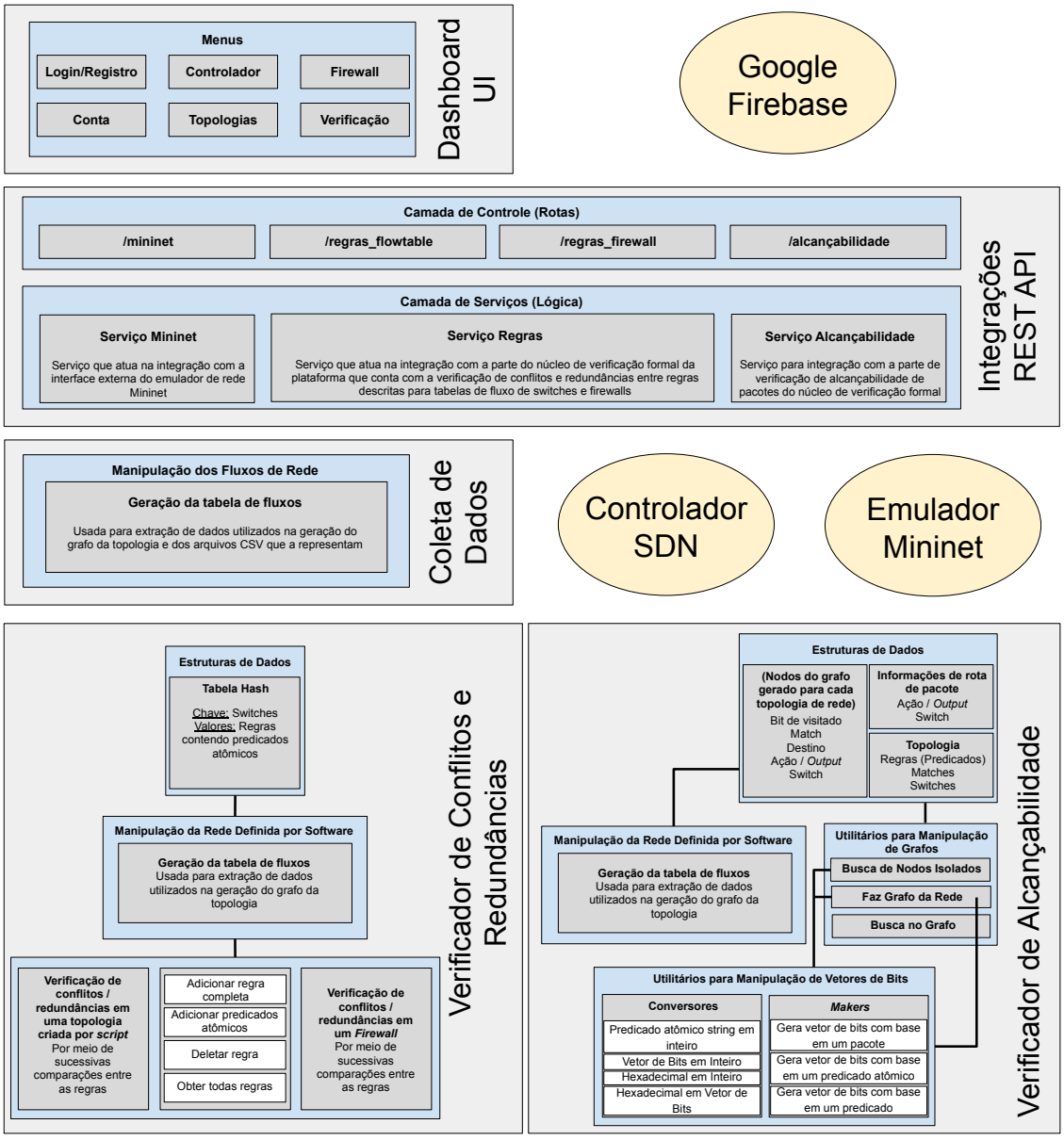
O módulo que faz a verificação de conflitos e redundâncias utiliza uma tabela *hash* em memória como estrutura de dados principal. Essa tabela *hash* só é obtida após a manipulação dos fluxos de rede ter sido completada pelo componente de coleta de dados. Na tabela *hash* utilizada, cada chave representa um switch presente na rede. Ademais, os valores são as regras contendo predicados atômicos no formato strings hexadecimais. Além disso, o módulo possui métodos para gerenciamento de regras tanto em switches quanto em firewalls. Dentre esses métodos, destacam-se as funcionalidades de (i) adicionar regra completa, (ii) adicionar o predicado atômico, (iii) deletar regra, e (iv) obter todas as regras de uma tabela de encaminhamento de algum switch ou de todas as regras presentes

em um firewall. Para executar essas funções internamente, o módulo se comunica diretamente com o controlador SDN Floodlight, utilizando o componente de integração REST API, o qual já foi apresentado anteriormente nesse trabalho.

Já o módulo que faz a verificação de alcançabilidade possui grafo como estrutura de dado principal. Dentro do módulo, cada topologia é modelada como um grafo de fluxos na rede. Cada nodo desse grafo possui informações referentes a cada fluxo, no formato de vetores de bits. O módulo possui internamente funções para montagem do grafo de rede e para busca em largura no grafo. Além disso, o módulo possui internamente diversos métodos de manipulação de vetores de bits, os quais são utilizados pelos métodos que executam partes do algoritmo de verificação. Os métodos de manipulação se dividem em dois grupos: (i) conversores (ii) *makers*. Nos métodos conversores se encontram os métodos que fazem conversões entre tipos de variáveis dentro da plataforma (e.g., hexadecimal em vetor de bits). Já no grupo de *makers*, encontram-se os métodos para gerações de vetores de bits a partir de alguma informação presente em algum fluxo ou na rede, como, por exemplo, dados de reconhecimento de um pacote.

Por fim, vale ressaltar que os dois verificadores não se comunicam entre si, sendo ambos acessados separadamente via componente de integração de serviços via REST API. A Figura 3.6 ilustra como os componentes da plataforma estão organizados entre si, sendo as circunferências em amarelo os componentes externos. Ademais, na figura, a parte mais inferior representa o menor nível de abstração dentro da Plataforma Placidus.

Figura 3.6: Estruturação dos componentes internos dentro do Placidus



Fonte: O Autor

4 PROTÓTIPO E RESULTADOS EXPERIMENTAIS

Neste capítulo é apresentada da implementação do protótipo da Plataforma Placidus[1] e seus resultados experimentais. A Seção 4.1 apresenta a implementação do dashboard da plataforma, situado no lado do cliente da mesma. Além disso, nessa seção, é apresentada a organização dos serviços da plataforma e como a plataforma cumpre com os requisitos de análise prévia para ser uma plataforma disponível de qualquer lugar.

Já a Seção 4.2 apresenta os resultados experimentais obtidos com a Plataforma Placidus. Dentro dessa seção são mostrados os cenários de teste e os desempenhos obtidos com os verificadores formais presentes na plataforma.

4.1 Implementação da Plataforma de Gerenciamento de Redes com Verificação Formal para SDN

Um dos principais propósitos da Plataforma Placidus é da mesma ser de fácil utilização e de poder ser acessada de qualquer lugar, em diferentes dispositivos. A implementação do Placidus é centrada nesses dois pontos, fornecendo uma interface adaptável para diferentes tipos de aparelhos e acessível em uma interface web online.

Além disso, a Plataforma Placidus possui um controle de usuários. Dessa forma, cada operador tem acesso apenas aos seus dados. Dentre os dados referentes a cada usuário estão (i) as informações de conta, (ii) informações das topologias criadas e em execução, e (iii) histórico de verificações formais executadas dentro da plataforma.

4.1.1 Arquitetura e Implantação do Dashboard

A arquitetura proposta para o dashboard implementado para a Plataforma Placidus é modular e escalável, assim como os serviços presentes no *back end*. A Figura 4.1 apresenta como a arquitetura proposta para o dashboard está modularizada dentro da plataforma descrita nesse trabalho.

O módulo *Common* contém as estruturas comuns a todos os outros módulos, tais como as estruturas de controle de acesso a menus e gerenciamento de autenticação. O módulo *Routes*, por sua vez, é responsável pela integração em tempo real com as informações

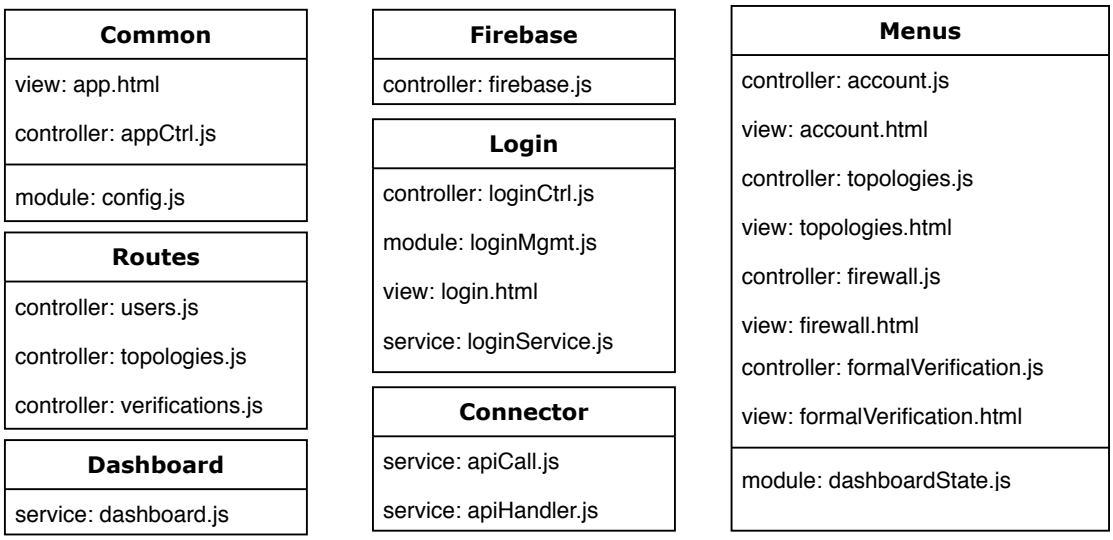
[1] A plataforma está disponível para acesso no link http://dawntech.brazilsouth.cloudapp.azure.com:8087

providas pelo banco de dados. Além disso, o mesmo é responsável pela atualização das *views* (visões) disponíveis para acesso dos usuários na plataforma. Já o módulo *Login* tem as funções necessárias para controle de acesso, assim como às visões necessárias para o registro de novos usuários e entrada dos mesmos dentro da plataforma.

O módulo *Dashboard* possui o controle de estados de todos os menus da plataforma. Além do mais, esse módulo contém o controle das configurações utilizadas dentro do dashboard, como estilo das páginas e controle de visualização para diferentes tipos de dispositivos. Já o módulo *Firebase* é responsável pela integração com o armazenamento no banco de dados utilizado. Ademais, esse módulo tem como função o controle do sistema de autenticação junto à plataforma Google Firebase.

O módulo *Connector*, por sua vez, possui todos os métodos e chamadas para a API REST dos serviços do *back end* da Plataforma Placidus. Por fim, o módulo *Menus* contém os controladores e as *views* necessárias para apresentar todas as informações disponíveis aos usuários logados na plataforma. Além disso, possui as funções necessárias para o uso das funcionalidades presentes na plataforma, como o gerenciamento de topologias e execução de verificação formal de propriedades.

Figura 4.1: Diagrama com a arquitetura do dashboard da Plataforma Placidus



Fonte: O Autor

4.1.2 Detalhes de Implementação

A Plataforma Placidus é heterogênea no que se diz respeito a linguagens de programação e frameworks e utilizados. No front-end da plataforma, é utilizado JavaScript com o uso do framework AngularJS[2] para a montagem dos controladores e renderização dos arquivos de visões HTML. Já para a personalização das páginas HTML do dashboard, optou-se por CSS3 com Bootstrap v3.3.7[3]. Além disso, todos os ícones utilizados nos menos da plataforma são providos pelo conjunto de ferramentas de fontes e ícones do projeto Font Awesome v4.7.0[4].

Ademais, todos os alertas da plataforma fazem uso do pacote SweetAlert[5], o qual permite o uso de pop-ups personalizados dentro do dashboard. Já para fazer o acesso em tempo real aos valores contidos no Google Firebase, utilizou-se o Framework AngularFire[6], o qual permite a dupla amarração entre os valores contidos no banco de dados e no escopo do dashboard. Atualmente 14 pacotes externos estão integrados ao dashboard da plataforma a fim de auxiliar no desenvolvimento da mesma. Sem contar os códigos de terceiros, foram implementadas, ao longo desse trabalho, 3397 linhas de código no front-end da Plataforma Placidus. Os módulos principais da aplicação são responsáveis por 56,4%, os arquivos de estilo CSS por 32,7% e os arquivos de indexação HTML por 10,9% da base de código própria do dashboard.

Já no *back end* da plataforma, foi utilizada a linguagem Python 3.6 com a utilização do framework Flask[7]. Para permitir uma melhor separação entre os módulos na arquitetura do *back end*, optou-se pelo uso do módulo Blueprints, presente no framework Flask. Com o uso de blueprints, cada arquivo pode possuir rotas separadas, desde que em cada método seja adicionada uma anotação como a que segue no bloco a seguir, que foi utilizada na rota de adicionar uma topologia.

$$@mininet\_route.route(”mininetstart”, methods = [”POST”])$$

No *back end* do Placidus, onde ficam as rotas e serviços necessários para o gerenciamento de topologias e verificações formais, chegou-se ao número de 1190 linhas desenvolvidas exclusivamente para esse trabalho. Para calcular o número de linhas de-

[2]https://angularjs.org
[3]http://getbootstrap.com
[4]https://fontawesome.com
[5]https://sweetalert.js.org
[6]https://firebaseopensource.com/projects/angular/angularfire2
[7]https://pypi.org/project/Flask

senvolvidas no lado do servidor do Placidus, utilizou-se o comando mostrado no bloco a seguir.

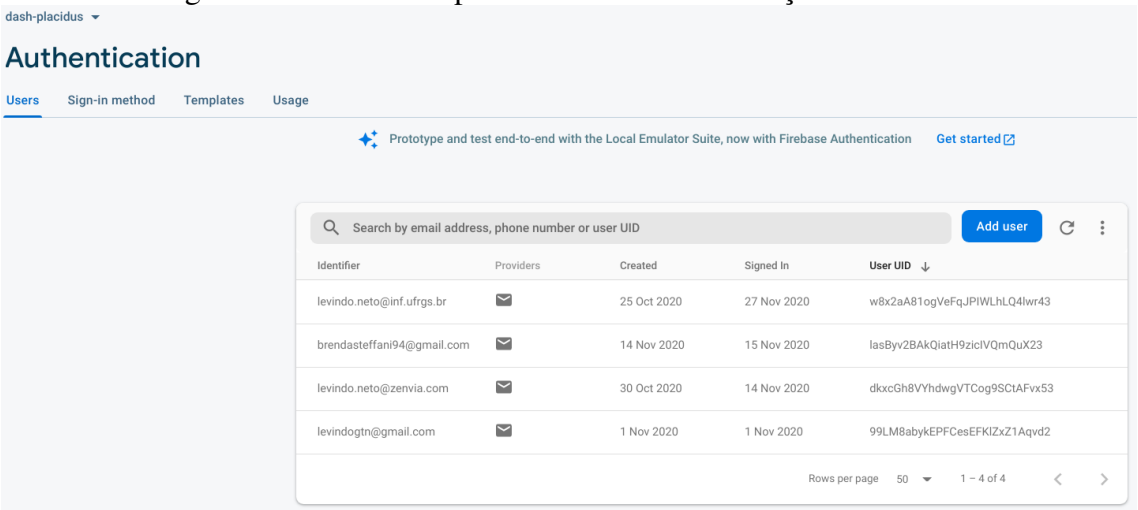
```
find rest/api/ -name '*.py' | xargs wc -l
```

4.1.3 Autenticação e Conta

Para se obter um controle de usuários e informações dentro da Plataforma Placidus, foi integrado dentro da mesma um sistema de autenticação. Além disso, a autenticação na plataforma descrita nesse trabalho é responsável pelo controle de acesso dos operadores às funcionalidades que o Placidus possui.

O sistema de autenticação dentro do Placidus é provido pelo Google Firebase, e o mesmo é integrado ao dashboard da plataforma. O Firebase provê um dashboard externo para controle dos usuários registrados na plataforma. Nesse dashboard, conforme visto na Figura 4.2, são fornecidas informações dos usuários que possuem registro no Placidus. As informações contidas são (i) chave de e-mail utilizada para identificação, (ii) data de criação da conta, (iii) data do último login, e (iv) o ID do usuário para uso interno.

Figura 4.2: Dashboard para controle de autenticação de usuários



Fonte: O Autor

Ademais, a plataforma do Google fornece um banco de dados em tempo real, o que permite com que usuários possam salvar seus dados pessoais, de seus históricos de verificações executadas e de suas topologias. Para acessar às funcionalidades do Placidus, o usuário deve criar uma conta na página de registro. As informações necessárias para o registro de conta na plataforma são (i) email, (ii) nome de usuário, e (iii) senha.

Após o registro e confirmação, o usuário pode logar na plataforma, utilizando o email e a senha cadastrados. Após entrar na plataforma, o usuário é automaticamente redirecionado para a página de conta. Dentro dessa página é possível conferir os dados de cadastro. Além disso, dentro da página de conta, é possível alterar o nome de usuário e a foto de perfil do usuário.

4.1.4 Controlador SDN e Firewall

Dentro do Placidus é possível ter acesso a informações da topologia de rede disponibilizadas pelo controlador SDN. Além disso, dentro da plataforma, é possível fazer o gerenciamento do firewall utilizado na rede.

O controlador SDN dentro do Placidus fornece informações completas sobre os switches e hosts presentes na topologia. Além disso, fornece dados referentes às conexões entre os dispositivos da topologia. Já no menu Firewall, é possível gerenciar status e regras do dispositivo firewall presente na topologia.

4.1.4.1 Controlador SDN

O controlador SDN no Placidus, o qual pode ser acessado ao clicar na opção *Controller* do menu lateral, é responsável pelo gerenciamento do plano de controle das topologias implantadas na plataforma. Nesse menu é possível obter informações de (i) switches, (ii) hosts, (iii) *links* (conexões) entre equipamentos de rede, (iv) status e (v) memória interna consumida pelo controlador.

Tanto o submenu Hosts, quanto o menu Links, possuem apenas tabelas informativas, porém sem ações disponíveis. O menu Hosts apresenta uma tabela de hosts conectados na topologia de rede atual. O submenu Switches, por sua vez, apresenta duas tabelas com informações referentes aos switches conectados e os seus papéis dentro da topologia. Cada switch pode ser acessado individualmente ao clicar no seu ID dentro da tabela de switches conectados. Assim, é aberta a página de detalhes de um switch, a qual apresenta informações mais detalhadas do equipamento e dos fluxos referentes a sua tabela de encaminhamento.

Já no submenu de gerenciamento de firewall, é possível fazer a ativação e desativação do firewall da rede virtual atual. Além disso, é possível alterar a máscara de sub-rede do firewall utilizado. Ademais, nesse submenu, é possível visualizar e adicionar regras

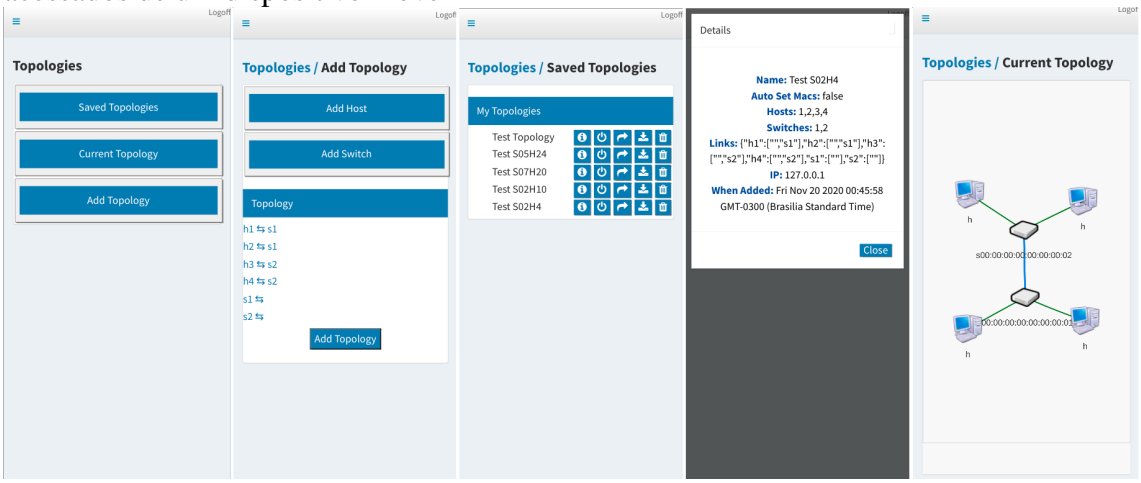
com ações de permissão ou bloqueio de pacotes na rede.

4.1.5 Gerenciamento de Topologias

O gerenciamento de topologias dentro do Placidus é separado em em três submenus: (i) criação de nova topologia, (ii) visualização da topologia atual, e (iii) listagem de todas topologias do usuário. Na criação de topologias, o usuário pode optar em montar uma topologia manualmente, explicitando a quantidade de switches, hosts e as conexões entre os dispositivos.

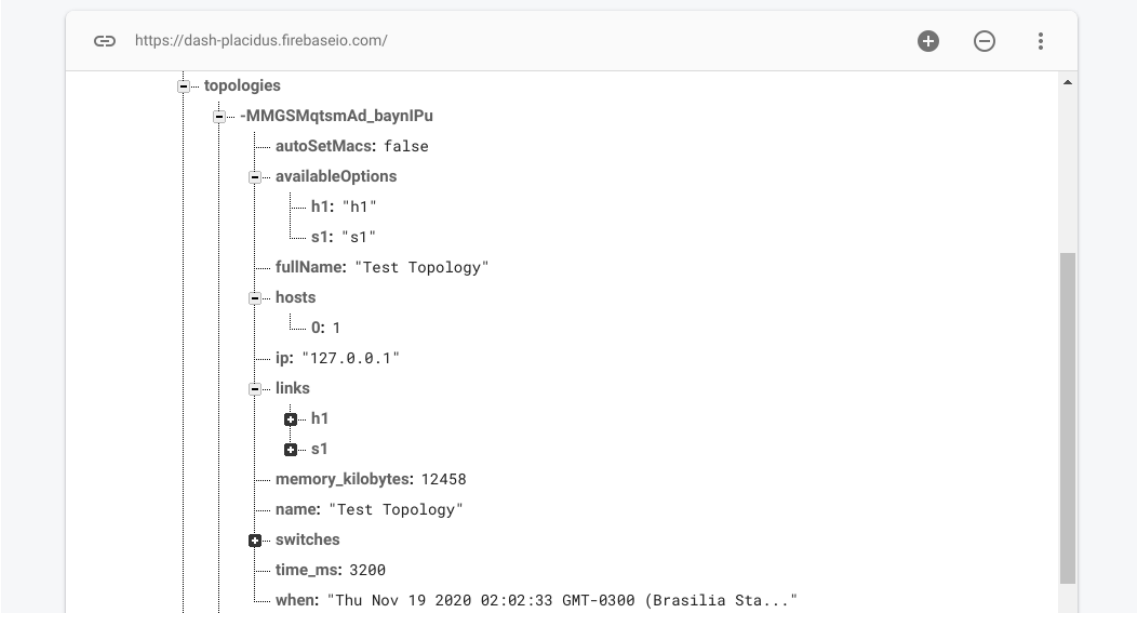
Além disso, ou usuário tem a opção de criar topologias automáticas, informando apenas a quantidade de hosts e switches. Nesse último caso, o serviço de criação de topologias do Placidus faz automaticamente a conexão de todos os dispositivos entre si, o que leva a uma quantidade de conexões (*links*) igual a $(\#switches + \#hosts) \cdot (\#switches + \#hosts - 1)$. Além disso, cada topologia registrada pode ser implantada e depois visualizada no menu de topologia atual. Por fim, todas as topologias adicionadas são sincronizadas no banco de dados da plataforma, podendo ser acessadas do menu de topologias salvas. A Figura 4.4 mostra como uma topologia é modelada no banco de dados de tempo real do Google Firebase quando adicionada via gerenciador de topologias. Nota-se que cada topologia possui um *hash* de identificação diferente, e que todas as topologias estão em um nível imediatamente abaixo da chave de usuários dentro do banco de dados.

Figura 4.3: Submenus de gerenciamento de topologias dentro da Plataforma Placidus acessados de um dispositivo móvel



Fonte: O Autor

Figura 4.4: Exemplo de topologia salva no banco de dados utilizado pelo dashboard da Plataforma Placidus



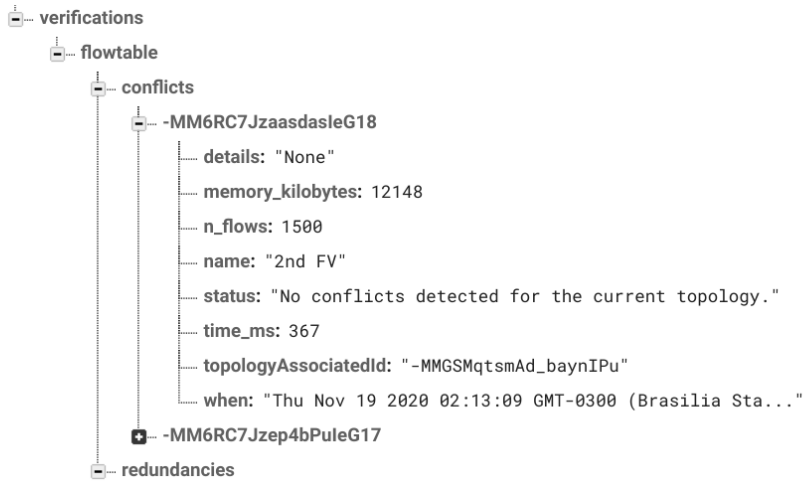
Fonte: O Autor

4.1.6 Menus de Verificação Formal

Como foi frisado anteriormente nesse trabalho, um dos maiores propósitos da Plataforma Placidus é a facilitação no gerenciamento de diferentes tarefas no contexto de redes definidas por software. O mesmo acontece para execução de verificações formais de propriedades dentro das topologias de rede implantadas na plataforma.

Ao invés de usuário ter que rodar scripts manualmente para fazer verificações, e depois coletar os dados separadamente, a plataforma as disponibiliza de maneira simples e intuitiva dentro dos submenus de verificação. Atualmente, o usuário pode, no dashboard da plataforma, verificar regras (conflitos e redundâncias) em tabelas de encaminhamento dos switches da topologia atual e no firewall da rede virtual implantada. Além disso, o usuário pode fazer a verificação de alcançabilidade dentro da topologia atual. Após as verificações serem executadas, o histórico das mesmas são salvos no banco de dados e ficam acessíveis aos usuários dentro da plataforma. Dentre as informações salvas no histórico de verificações, destacam-se (i) tempo de execução, (ii) memória utilizada, (iii) quantidade de fluxos ou dispositivos, e (iv) topologia testada. A modelagem das verificações no banco de dados utilizado, em um nível abaixo da chave de usuários, pode ser observada na Figura 4.5.

Figura 4.5: Exemplo de verificação sincronizada no Google Firebase integrado ao Placidus



Fonte: O Autor

4.2 Resultados experimentais

Nessa seção são apresentados os resultados experimentais dos módulos do núcleo de verificação da Plataforma Placidus. As propriedades avaliadas para esses módulos são tempo de execução e memória utilizada. Esses dados são avaliados de acordo com o tamanho de cada topologia e quantidade de fluxos contidos na mesma. Para cada ponto presente nos gráficos de resultados, uma sequência de 10 execuções foi executada. No eixo Y dos gráficos, colocaram-se as médias aritméticas dos resultados em cada ponto dessas execuções. Ademais, barras verticais foram adicionadas para mostrar a diferença entre o maior e o menor valor obtido para cada sequência.

Todas as topologias utilizadas para os experimentos realizados foram montadas dentro do gerenciador de topologias do Placidus, o qual está integrado com emulador de rede Mininet. Além disso, depois da criação das redes virtuais, um script de inicialização próprio foi utilizado para introduzir fluxos dentro das topologias. A plataforma, para os testes executados, foi implantada em uma máquina virtual na provedora de nuvem Microsoft Azure. A máquina virtual criada teve as seguintes configurações: memória RAM de 28 GB, 4 vCPU com e sistema operacional Ubuntu Server 18.04 LTS Gen 1.

4.2.1 Resultados dos Testes de Verificação de Regras

Para avaliar o tempo de execução do núcleo de verificação de conflitos e redundâncias do Placidus, mediu-se a diferença de tempo entre o início da coleta de dados do controlador SDN e o final da verificação executada no módulo. A Figura 4.6 apresenta um gráfico gerado com a medição de tempo de verificação de regras ao variar a quantidade de fluxos de rede de 1500 a 18000. A quantidade de fluxos de rede foi aumentada a partir da criação de um número diferente de switches, hosts e conexões entre esses dispositivos com a utilização do gerenciador de topologias do Placidus. Os testes de memória utilizada também foram executados fazendo avaliação de fluxos dentro das topologias de rede virtual.

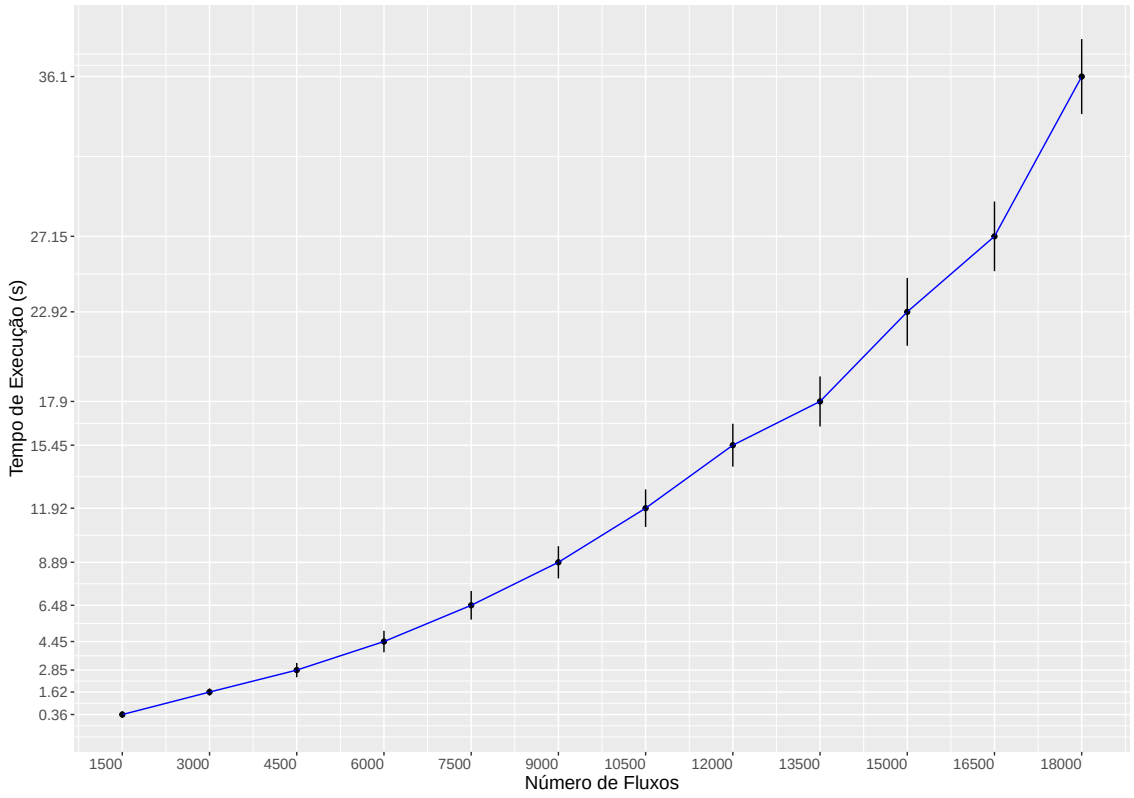
O gráfico da Figura 4.7 apresenta os resultados do uso de memória para cada número de fluxos contidos na rede. Os resultados encontrados tanto para a quantidade de tempo quanto para a memória utilizada pelo verificador estão de acordo com a complexidade de tempo do algoritmo implementado, que é $O(n^2)$. Pode-se notar, nos dois gráficos gerados, um comportamento quadrático, o que corrobora para a complexidade temporal do algoritmo utilizado nesse módulo de verificação. Esse comportamento quadrático deverá ser melhorado a partir do momento que acontecer a uniformização das estruturas de dados para vetores de bits dentro da Plataforma Placidus. Isso permitirá o uso de outros algoritmos para executar as verificações de conflitos e redundância entre regras de maneira mais eficiente.

4.2.2 Resultados dos Testes de Verificação de Alcançabilidade

Para os testes do verificador de alcançabilidade, foi-se alterando as topologias de maneira a obter diferentes números de dispositivos e conexões para executar a verificação da propriedade em questão. Dentro dessas topologias criadas, alterou-se a quantidade de dispositivos de 10 a 140. Ademais, as topologias foram criadas de maneira aleatória dentro do gerenciador de topologias do dashboard da Plataforma Placidus. Assim como na análise do verificador de regras, para cada ponto no gráfico houve a execução de uma sequência de 10 testes, marcando no eixo Y a média aritmética dos resultados obtidos para cada ponto, assim como a marcação de barras de erro em cada ponto.

Como visto anteriormente, a complexidade de tempo do verificador de alcançabilidade é baseada na quantidade de vértices e arestas do grafo que representa a topologia.

Figura 4.6: Tempo para fazer a verificação de conflitos e redundâncias entre regras dentro do Placidus.

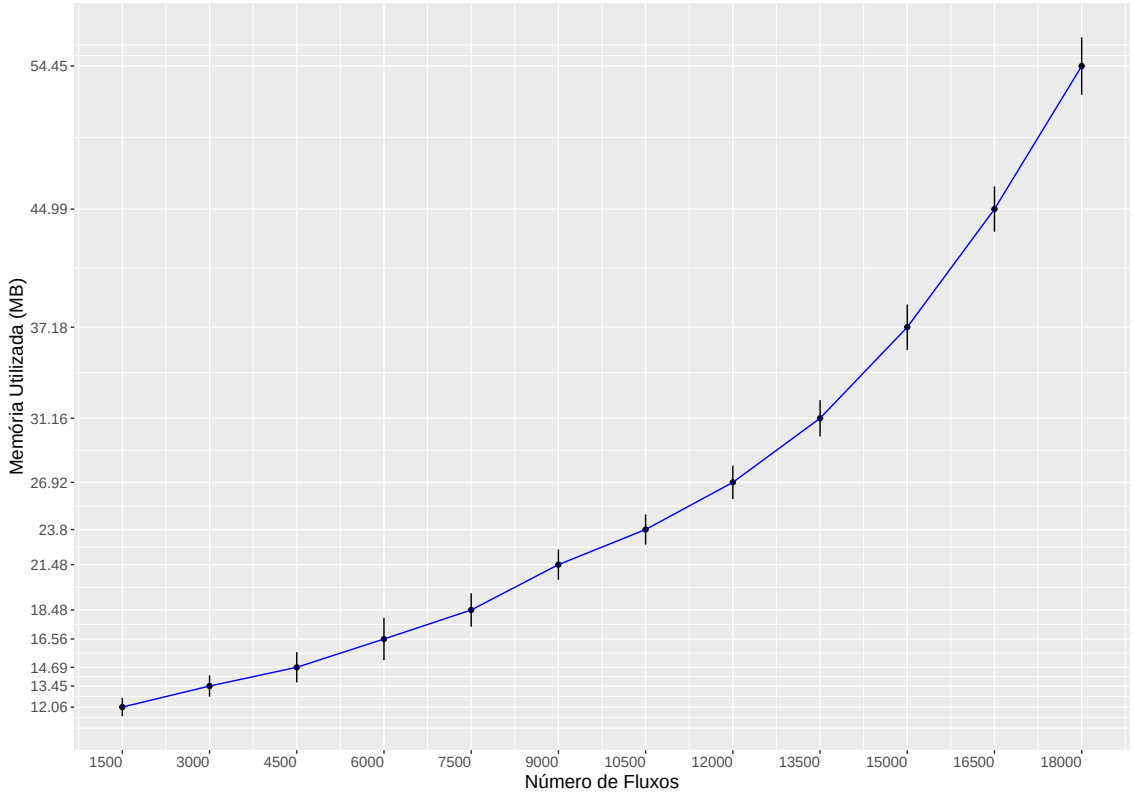


Fonte: O Autor

Nas topologias de rede virtual criadas no Placidus, os dispositivos representam os vértices, e as conexões entre os mesmos representam as arestas. Assim como no modo de verificação de regras, o gráfico gerado nesse módulo para o tempo de execução e memória utilizada convergiu para um formato de reta, o que está de acordo com a complexidade do algoritmo utilizado.

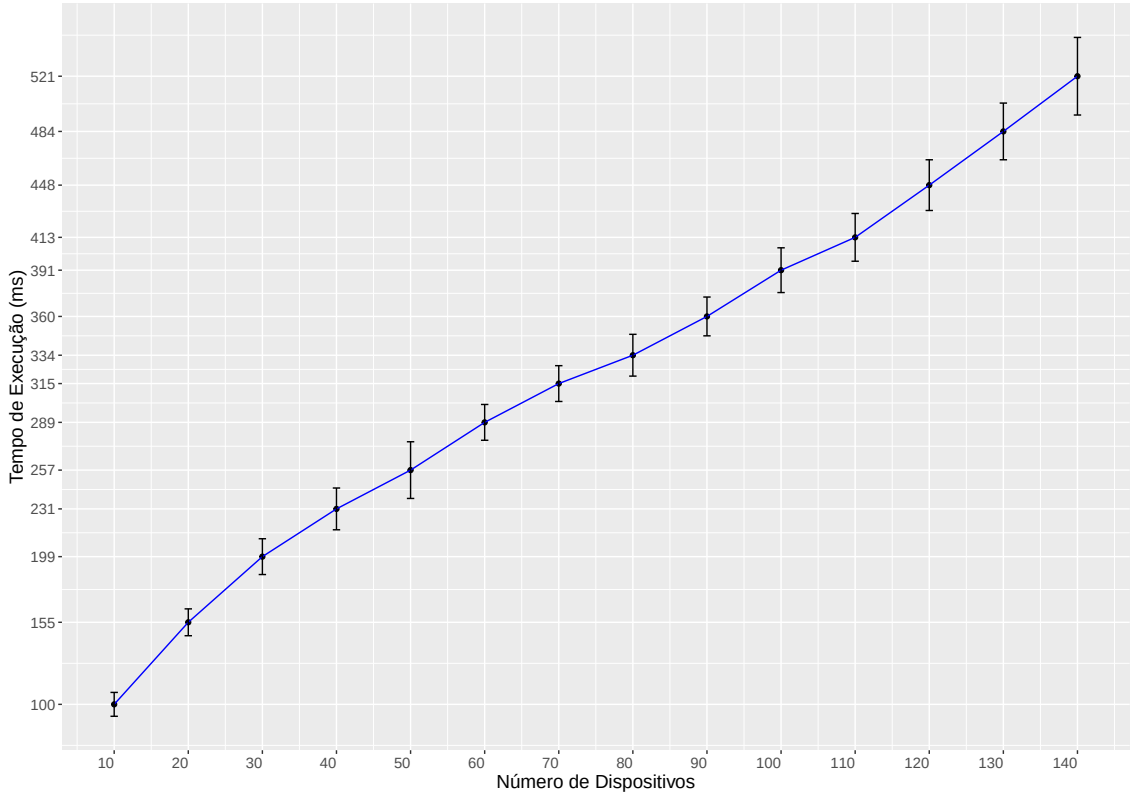
A Figura 4.8 apresenta os resultados obtidos com o verificador para verificar se um pacote dentro de um dispositivo *a* da topologia consegue chegar em outro dispositivo *b* dentro da mesma. Pode-se notar um tempo de 100 milissegundos para verificar a menor topologia testada (6 hosts e 4 switches). Esse tempo leva em consideração a obtenção dos dados do controlador feita pelo módulo de coleta de dados do Placidus. A Figura 4.9 apresenta a quantidade de recursos memória (em MB) utilizada nos mesmos testes. Assim como no tempo de execução, o uso de memória se manteve linear à medida que o número de dispositivos e conexões na topologia aumentaram. Isso corrobora com o desempenho e complexidade linear do verificador de alcançabilidade da Plataforma Placidus.

Figura 4.7: Memória utilizada para fazer a verificação de conflitos e redundâncias entre regras dentro do Placidus.



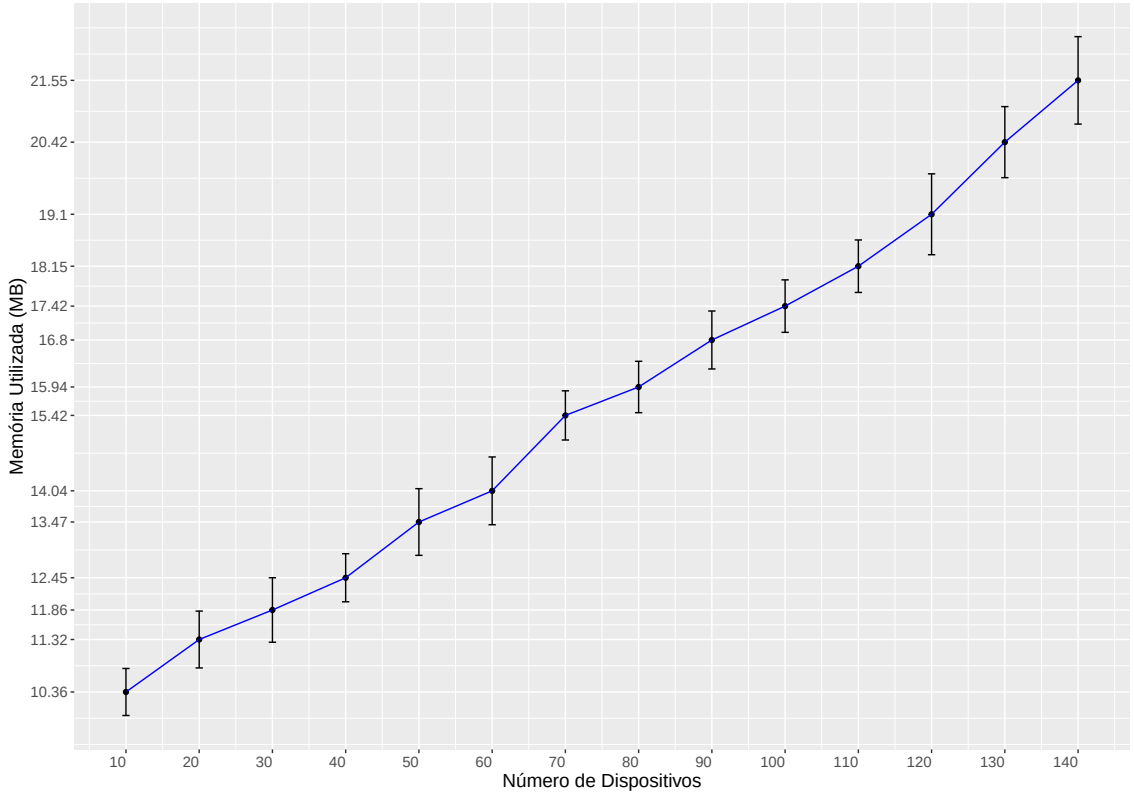
Fonte: O Autor

Figura 4.8: Tempo para fazer a verificação de alcançabilidade dentro do Placidus.



Fonte: O Autor

Figura 4.9: Memória utilizada para fazer a verificação de alcançabilidade dentro do Placidus.



Fonte: O Autor

5 CONCLUSÃO

Nesse trabalho foi proposta uma plataforma para gerenciamento e verificação formal de propriedades para Redes Definidas por Software, o Placidus. Após uma introdução, foram revisados conceitos da arquitetura de SDN e técnicas de verificação em redes de computadores.

Além disso, foram apresentadas plataformas de gerenciamento de redes que possuem similaridades com a plataforma desenvolvida nesse trabalho. Na sequência, foi apresentada a arquitetura da plataforma e as técnicas de verificação utilizadas no núcleo de verificação da mesma. Por fim, o protótipo desenvolvido para plataforma e os resultados da mesma foram discutidos.

5.1 Contribuições

As contribuições feitas pela plataforma desenvolvida nesse trabalho são na área de gerenciamento e na área de verificação formal de redes definidas por software. Na área de gerenciamento de SDN, o Placidus fornece uma plataforma acessível de qualquer lugar e com controle de acesso para que os seus usuários possam gerenciar suas topologias e suas verificações de maneira simples e efetiva, assim como manter histórico das atividades realizadas na plataforma.

Além disso, dentro da plataforma é possível fazer o gerenciamento do controlador SDN utilizado, podendo acessar as informações dos dispositivos da rede e adicionar regras nos mesmos. Ademais, o Placidus torna mais fácil a implantação de topologias de teste por pesquisadores e operadores de rede, uma vez que a plataforma fornece um gerenciador para criação e manipulação de topologias de redes virtuais. Nesse gerenciador fornecido pela plataforma, é possível adicionar hosts, switches e *links* (conexões) entre esses dispositivos diretamente em um dashboard integrado com os serviços desenvolvidos para conectar com o emulador de rede Mininet.

Já na área de verificação formal de propriedades em SDN, é fornecido nesse trabalho um framework de verificação formal embutido na plataforma desenvolvida. Ademais, o mesmo possui atualmente verificação de conflitos e redundâncias de regras para tabelas de encaminhamento de switches e para firewalls. Além disso, o núcleo de verificação tem o módulo de verificação de alcançabilidade para topologias de rede virtuais. O framework desenvolvido dentro do módulo de verificação da Plataforma Placidus é modular escalá-

vel, permitindo assim a sua expansão de maneira simplificada, o que faz com que a adição de novos tipos de verificação em trabalhos futuros seja mais rápida e fácil.

5.2 Trabalhos Futuros

Trabalhos futuros incluem expansões na plataforma desenvolvida. Essas expansões são tanto no dashboard da plataforma quanto no núcleo de verificação. No que tange o dashboard do Placidus, trabalhos futuros incluem (i) melhorias de usabilidade na versão móvel da plataforma, (ii) notificações de alertas sobre as topologias, (iii) geração de relatórios personalizados, e (iv) agendamento de eventos de verificações.

Já no núcleo da verificação formal, trabalhos futuros abrangem a exploração de outras propriedades globais de redes. Dentre essas propriedades, destacam-se a detecção de *black holes* (verificar se existe sumidouros de pacotes na rede), detecção de *loops*, e a verificação de *slice isolation* dentro de topologias. Além disso, para os próximos passos dentro do Placidus, pretende-se aprimorar os algoritmos já propostos e uniformizar as estruturas de dados para os serviços desenvolvidos. Por fim, intenta-se deixar a plataforma mais robusta para poder suportar uma quantidade maior de usuários simultâneos sem que haja queda em sua fluidez e desempenho.

REFERÊNCIAS

AGBORUBERE, B.; SANCHEZ, E. Openflow communications and tls security in software-defined networks. In: . [S.l.: s.n.], 2017. p. 560–566.

Ahmad, I. et al. Security in software defined networks: A survey. **IEEE Communications Surveys Tutorials**, v. 17, n. 4, p. 2317–2346, 2015.

Ahmad, S. et al. Novel approach using deep learning for intrusion detection and classification of the network traffic. In: **2020 IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA)**. [S.l.: s.n.], 2020. p. 1–6.

Al-Haj, S.; Tolone, W. J. Flowtable pipeline misconfigurations in software defined networks. In: **2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)**. [S.l.: s.n.], 2017. p. 247–252. ISSN null.

Anderson, J.; Cho, J. Software defined network based virtual machine placement in cloud systems. In: **MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM)**. [S.l.: s.n.], 2017. p. 876–881.

Asadollahi, S. et al. Scalability of software defined network on floodlight controller using ofnet. In: **2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT)**. [S.l.: s.n.], 2017. p. 1–5.

BALL, T. et al. Vericon: Towards verifying controller programs in software-defined networks. **SIGPLAN Not.**, ACM, New York, NY, USA, v. 49, n. 6, p. 282–293, jun. 2014. ISSN 0362-1340. Disponível em: <http://doi.acm.org/10.1145/2666356.2594317>.

Bertacco, V.; Olukotun, K. Efficient state representation for symbolic simulation. In: **Proceedings 2002 Design Automation Conference (IEEE Cat. No.02CH37324)**. [S.l.: s.n.], 2002. p. 99–104.

Caiazza, C. et al. Mecperf: An application-level tool for estimating the network performance in edge computing environments. In: **2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)**. [S.l.: s.n.], 2020. p. 1163–1168.

Collings, J.; Liu, J. An openflow-based prototype of sdn-oriented stateful hardware firewalls. In: **2014 IEEE 22nd International Conference on Network Protocols**. [S.l.: s.n.], 2014. p. 525–528.

da Silva, A. S.; Schaeffer-Filho, A. Armor: An architecture for diagnosis and remediation of network misconfigurations. In: **2019 IEEE Symposium on Computers and Communications (ISCC)**. [S.l.: s.n.], 2019. p. 1–6.

Dahl, G. E. et al. Large-scale malware classification using random projections and neural networks. In: **2013 IEEE International Conference on Acoustics, Speech and Signal Processing**. [S.l.: s.n.], 2013. p. 3422–3426.

Fang, Y.; Lu, Y. Real-time verification of network properties based on header space. **IEEE Access**, v. 8, p. 36789–36806, 2020.

Farhadi, H.; Du, P.; Nakao, A. Enhancing openflow actions to offload packet-in processing. In: **The 16th Asia-Pacific Network Operations and Management Symposium**. [S.l.: s.n.], 2014. p. 1–6.

FEAMSTER, N.; BALAKRISHNAN, H. Detecting bgp configuration faults with static analysis. In: **Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2**. Berkeley, CA, USA: USENIX Association, 2005. (NSDI'05), p. 43–56. Disponível em: <http://dl.acm.org/citation.cfm?id=1251203.1251207>.

FEAMSTER, N.; REXFORD, J.; ZEGURA, E. The road to sdn: An intellectual history of programmable networks. **SIGCOMM Comput. Commun. Rev.**, Association for Computing Machinery, New York, NY, USA, v. 44, n. 2, p. 87–98, abr. 2014. ISSN 0146-4833. Disponível em: <https://doi.org/10.1145/2602204.2602219>.

FOUNDATION, O. N. **OpenFlow Switch Specification documentation**. 2012. <https://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.3.1.pdf>. Accessed: 2019-11-29.

Garrich, M. et al. It and multi-layer online resource allocation and offline planning in metropolitan networks. **Journal of Lightwave Technology**, v. 38, n. 12, p. 3190–3199, 2020.

Ge, X. et al. Dyta: dynamic symbolic execution guided with static verification results. In: **2011 33rd International Conference on Software Engineering (ICSE)**. [S.l.: s.n.], 2011. p. 992–994.

Gobrial, M. N. Evaluation of border gateway protocol (bgp) version 4 (v4) in the tactical environment. In: **Proceedings of MILCOM '96 IEEE Military Communications Conference**. [S.l.: s.n.], 1996. v. 2, p. 490–495 vol.2.

GORANSSON, P.; BLACK, C. **Software Defined Networks: A Comprehensive Approach**. 1st. ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2014. ISBN 012416675X, 9780124166752.

Hamaguchi, K. Symbolic simulation heuristics for high-level design descriptions with uninterpreted functions. In: **Sixth IEEE International High-Level Design Validation and Test Workshop**. [S.l.: s.n.], 2001. p. 25–30.

Hamed, H.; Al-Shaer, E.; Marrero, W. Modeling and verification of ipsec and vpn security policies. In: **13TH IEEE International Conference on Network Protocols (ICNP'05)**. [S.l.: s.n.], 2005. p. 10 pp.–278.

HSIEH, Y. W.; LEVITAN, S. P. Model abstraction for formal verification. In: **Proceedings Design, Automation and Test in Europe**. [S.l.: s.n.], 1998. p. 140–147.

Hu, F.; Hao, Q.; Bao, K. A survey on software-defined network and openflow: From concept to implementation. **IEEE Communications Surveys Tutorials**, v. 16, n. 4, p. 2181–2206, 2014.

Ismail, M. N.; Syarmila, S. Network management system framework and development. In: **2009 International Conference on Future Computer and Communication**. [S.l.: s.n.], 2009. p. 450–454.

Ivutin, A. N.; Voloshko, A. G. Method of formal verification of program code based on petri net with additional semantic relations. In: **2020 ELEKTRO**. [S.l.: s.n.], 2020. p. 1–6.

Jarraya, Y.; Madi, T.; Debbabi, M. A survey and a layered taxonomy of software-defined networking. **IEEE Communications Surveys Tutorials**, v. 16, n. 4, p. 1955–1980, 2014.

Kang, M. et al. Formal modeling and verification of sdn-openflow. In: **2013 IEEE Sixth International Conference on Software Testing, Verification and Validation**. [S.l.: s.n.], 2013. p. 481–482.

Kazaz, T. et al. One approach to the development of custom snmp agents and integration with management systems. In: **2012 Proceedings of the 35th International Convention MIPRO**. [S.l.: s.n.], 2012. p. 557–561.

KAZEMIAN, P.; VARGHESE, G.; MCKEOWN, N. Header space analysis: Static checking for networks. In: **Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation**. USA: USENIX Association, 2012. (NSDI'12), p. 9.

Kim, Y.; Kang, M. Formal verification of sdn-based firewalls by using tla+. **IEEE Access**, v. 8, p. 52100–52112, 2020.

Koike, E.; Nishizaki, S. Software analysis of internet bots using a model checker. In: **2013 International Conference on Information Science and Cloud Computing Companion**. [S.l.: s.n.], 2013. p. 242–245.

Kompella, R. R. et al. Detection and localization of network black holes. In: **IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications**. [S.l.: s.n.], 2007. p. 2180–2188.

Kreutz, D. et al. Software-defined networking: A comprehensive survey. **Proceedings of the IEEE**, v. 103, n. 1, p. 14–76, 2015.

Li, Y. et al. A survey on network verification and testing with formal methods: Approaches and challenges. **IEEE Communications Surveys Tutorials**, v. 21, n. 1, p. 940–969, 2019.

LIU, A. X. Formal verification of firewall policies. In: . East Lansing, MI 48824-1266, U.S.A.: [s.n.], 2005.

Ljubojević, M.; Bajić, A.; Mijić, D. Centralized monitoring of computer networks using zenoss open source platform. In: **2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH)**. [S.l.: s.n.], 2018. p. 1–5.

MAJUMDAR, R.; TETALI, S. D.; WANG, Z. Kuai: A model checker for software-defined networks. In: **Formal Methods in Computer-Aided Design (FMCAD), 2014**. [S.l.: s.n.], 2014. p. 163–170.

Markowski, M.; Ryba, P.; Puchała, K. Software defined networking research laboratory-experimental topologies and scenarios. In: **2016 Third European Network Intelligence Conference (ENIC)**. [S.l.: s.n.], 2016. p. 252–256.

MCKEOWN, N. **OpenFlow: Enabling Innovation in Campus Networks**. Volume 38, issue 1. [S.l.]: SIGCOMM, 2008.

MCKEOWN, N. et al. Openflow: Enabling innovation in campus networks. **SIGCOMM Comput. Commun. Rev.**, Association for Computing Machinery, New York, NY, USA, v. 38, n. 2, p. 69–74, mar. 2008. ISSN 0146-4833. Disponível em: <https://doi.org/10.1145/1355734.1355746>.

Mouradian, A.; Blum, I. A. Formal verification of real-time wireless sensor networks protocols: Scaling up. In: **2014 26th Euromicro Conference on Real-Time Systems**. [S.l.: s.n.], 2014. p. 41–50.

Nam, J. et al. Operator-defined reconfigurable network os for software-defined networks. **IEEE/ACM Transactions on Networking**, v. 27, n. 3, p. 1206–1219, 2019.

Narantuya, J. et al. Sdn-based ip shuffling moving target defense with multiple sdn controllers. In: **2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks – Supplemental Volume (DSN-S)**. [S.l.: s.n.], 2019. p. 15–16.

Panda, A. et al. Dynamic hard timeout based flow table management in openflow enabled sdn. In: **2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN)**. [S.l.: s.n.], 2019. p. 1–6.

PATHAK, K.; TIWARI, A.; CHAUDHARI, N. S. A reduction of 3-sat problem from optimal sanitization in association rule hiding. In: **2011 International Conference on Emerging Trends in Networks and Computer Communications (ETNCC)**. [S.l.: s.n.], 2011. p. 43–46.

Petruti, C. et al. Automatic management solution in cloud using ntopng and zabbix. In: **2018 17th RoEduNet Conference: Networking in Education and Research (RoEduNet)**. [S.l.: s.n.], 2018. p. 1–6.

Qadir, J.; Hasan, O. Applying formal methods to networking: Theory, techniques, and applications. **IEEE Communications Surveys Tutorials**, v. 17, n. 1, p. 256–291, Firstquarter 2015. ISSN 2373-745X.

RADOJICIC, C.; PURUSOTHAMAN, T.; GRIMM, C. Towards formal validation: Symbolic simulation of systemc models. In: **2015 10th International Conference on Design Technology of Integrated Systems in Nanoscale Era (DTIS)**. [S.l.: s.n.], 2015. p. 1–6.

Renita, J.; Elizabeth, N. E. Network's server monitoring and analysis using nagios. In: **2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)**. [S.l.: s.n.], 2017. p. 1904–1909.

Santos da Silva, A. et al. Atlantic: A framework for anomaly traffic detection, classification, and mitigation in sdn. In: **NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium**. [S.l.: s.n.], 2016. p. 27–35.

Satasiya, D.; Raviya Rupal D. Analysis of software defined network firewall (sdf). In: **2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)**. [S.l.: s.n.], 2016. p. 228–231.

SETHI, D.; NARAYANA, S.; MALIK, S. Abstractions for model checking sdn controllers. In: **2013 Formal Methods in Computer-Aided Design**. [S.l.: s.n.], 2013. p. 145–148.

Sgambelluri, A. et al. Openflow-based segment protection in ethernet networks. **IEEE/OSA Journal of Optical Communications and Networking**, v. 5, n. 9, p. 1066–1075, 2013.

Shaoying Liu. Formal verification of condition data flow diagrams for assurance of correct network protocols. In: **17th International Conference on Advanced Information Networking and Applications, 2003. AINA 2003.** [S.l.: s.n.], 2003. p. 289–292.

Sokappadu, B. et al. Software defined networks: Issues and challenges. In: **2019 Conference on Next Generation Computing Applications (NextComp)**. [S.l.: s.n.], 2019. p. 1–5.

Vinayakumar, R.; Soman, K. P.; Poornachandran, P. Evaluating shallow and deep networks for secure shell (ssh)traffic analysis. In: **2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)**. [S.l.: s.n.], 2017. p. 266–274.

Wang, S. et al. Unified software-defined online network experiment platform for campus education. In: **2016 International Conference on Networking and Network Applications (NaNA)**. [S.l.: s.n.], 2016. p. 299–302.

Wickboldt, J. A. et al. Software-defined networking: management requirements and challenges. **IEEE Communications Magazine**, v. 53, n. 1, p. 278–285, 2015.

Xiang, G.; Jianlin, Q. Formal description of network protocols using raise specification language. In: **2011 International Conference on Network Computing and Information Security**. [S.l.: s.n.], 2011. v. 1, p. 185–188.

XIE, G. G. et al. On static reachability analysis of ip networks. In: **Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.** [S.l.: s.n.], 2005. v. 3, p. 2170–2183 vol. 3. ISSN 0743-166X.

Yang, H.; Yang, Y.; Tu, Y. S3r5: A snapshot storage system based on row with rapid rollback, recovery and read-write. In: **2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)**. [S.l.: s.n.], 2019. p. 2111–2118. ISSN null.

Yeo, M. et al. Flow-based malware detection using convolutional neural network. In: **2018 International Conference on Information Networking (ICOIN)**. [S.l.: s.n.], 2018. p. 910–913.

Zerrik, S. et al. Towards a decentralized and adaptive software-defined networking architecture. In: **2014 International Conference on Next Generation Networks and Services (NGNS)**. [S.l.: s.n.], 2014. p. 326–329.

Zhang, Y. et al. Atomic predicates-based data plane properties verification in software defined networking using spark. **IEEE Journal on Selected Areas in Communications**, v. 38, n. 7, p. 1308–1321, 2020.