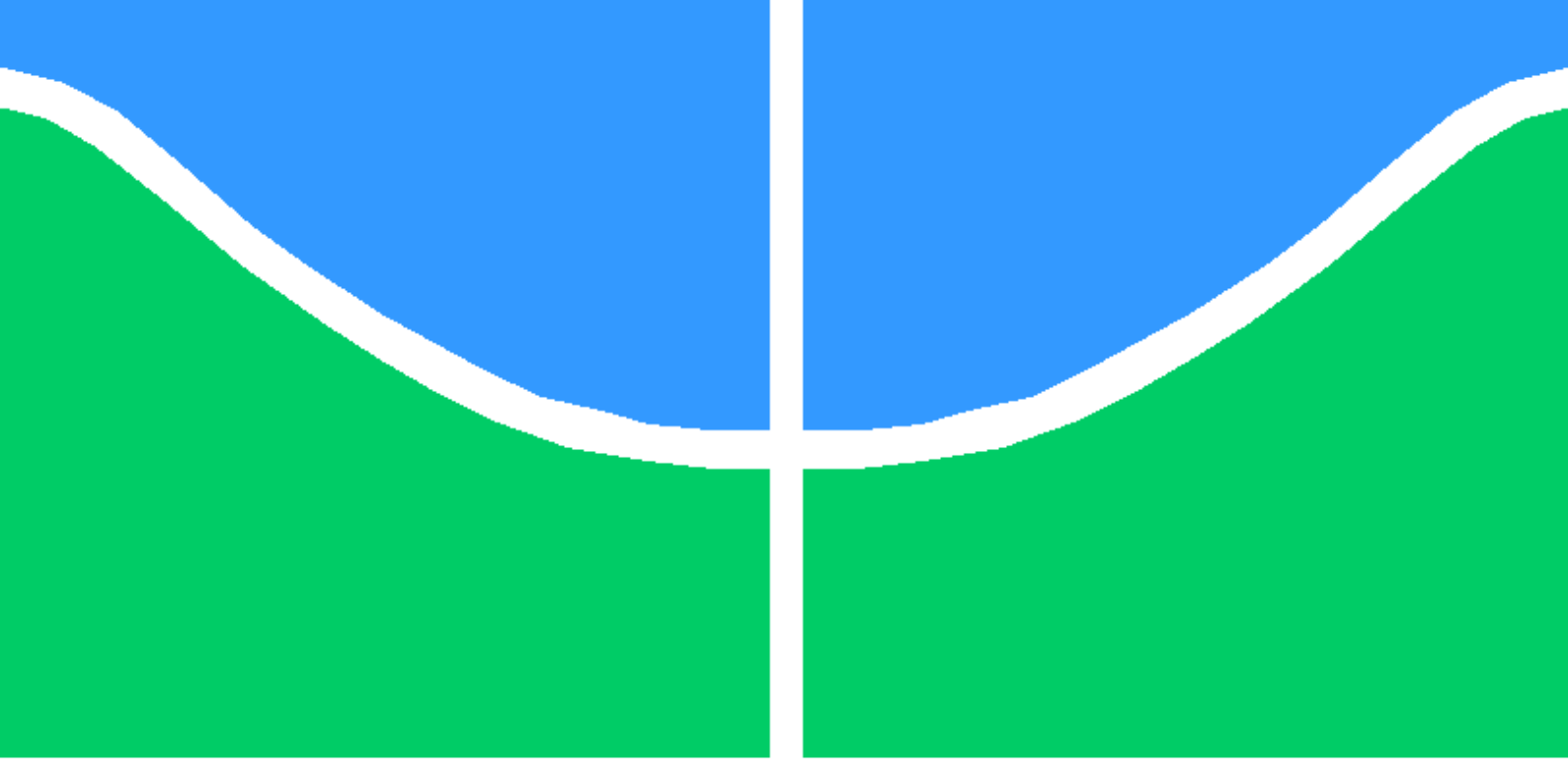

Implementação de arquitetura SDN-like para gerenciamento de dispositivos OpenWRT

Daniel Gomes Venzi Gonçalves

MONOGRAFIA DE GRADUAÇÃO EM ENGENHARIA DE REDES DE COMUNICAÇÃO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA

Implementação de arquitetura SDN-like para gerenciamento de dispositivos OpenWRT

Daniel Gomes Venzi Gonçalves

Relatório submetido ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Engenheiros de Redes de Comunicação.

Banca Examinadora:

Prof. Dr. Georges Daniel Amvame, ENE/UnB
Orientador

Prof. Dr. Fábio Lúcio Lopes de Mendonça, ENE/UnB
Examinador interno

Prof. Msc. Diego Martins de Oliveira, ENE/UnB
Examinador externo

BRASÍLIA, 30 DE SETEMBRO DE 2022.

Agradecimentos

É com extrema satisfação que agradeço a todos que estiveram em minha companhia durante todos esses anos na graduação de Engenharia de Redes de Comunicação. Gostaria de agradecer a minha família, que sempre me apoiou em minhas decisões, mesmo elas sendo ocasionalmente questionáveis, e por estarem sempre disponíveis durante minhas dificuldades com reservas inesgotáveis de amor e compreensão. Gostaria também de agradecer à minha avó, que, apesar de não estar mais presente, foi uma das maiores influências para o sucesso de minha trajetória até aqui.

Aos meus amigos do "Banco U", seu companheirismo e apoio durante os inúmeros desafios do curso foram essenciais para o meu sucesso acadêmico e, principalmente, para meu desenvolvimento pessoal. Fico extremamente contente em ver a pessoa que me tornei após todas as intempéries vivenciadas durante esses anos, tanto na minha vida pessoal, profissional e acadêmica, e me sinto inebriado por uma imensa gratidão por todas as experiências passadas, até mesmo por aquelas que me fizeram perder algumas noites de sono.

Ao corpo docente da Universidade de Brasília, que me proporcionou experiências acadêmicas multidisciplinares de alto nível, só posso expressar minha profunda gratidão pela oportunidade de ter feito parte dessa grande instituição. Em especial, gostaria de agradecer o professor Georges Daniel por toda a orientação e otimismo proporcionado durante o desenvolvimento deste trabalho.

Por fim, gostaria de agradecer a Deus, que, mesmo em nossa relação não convencional, esteve presente em meus momentos de dúvida e fraqueza.

Resumo

Diante dos avanços em tecnologias de rede como a abordagem de Software Defined Networks (SDN), se nota a possibilidade de empregar a programabilidade de redes advinda desse paradigma em diferentes entidades que podem compor uma rede empresarial ou residencial. Este trabalho consiste na apresentação e implementação de uma arquitetura para gerenciamento de dispositivos de ponto de acesso executando o sistema operacional OpenWRT a partir de uma entidade centralizadora que permite ações como listagem de dispositivos, regras e grupos lógicos, criação de grupos lógicos, segmentação de dispositivos gerenciados em grupos lógicos e criação e remoção de configurações suportadas. Com esse objetivo, serão descritas detalhadamente as funcionalidades das entidades que integram essa arquitetura e possibilitam o gerenciamento de pontos de acesso de uma rede (OpenWRT-SDN-Controller e OpenWRT-Daemon).

Após a implementação da arquitetura proposta, foram realizados testes para comprovar as funcionalidades descritas e demonstrar o funcionamento das entidades em uma rede simulada no software GNS3.

Espera-se que esse trabalho possa servir como uma base de implementação para uma solução mais robusta de gerenciamento de dispositivos OpenWRT.

Abstract

Given the advances in network technologies such as the Software Defined Networks (SDN) approach, it is noted the possibility of employing the network programmability coming from this paradigm in different entities that can compose a business or residential network. This work consists in the presentation and implementation of an architecture for managing access point devices running the OpenWRT operating system from a centralizing entity that allows actions such as listing devices, rules and logical groups, creating logical groups, segmenting managed devices into logical groups, and creating and removing supported configurations. With this objective, the functionalities of the entities that integrate this architecture and allow the management of a network's access points (OpenWRT-SDN-Controller and OpenWRT-Daemon) will be thoroughly described.

After the implementation of the proposed architecture, tests were performed to prove the described functionalities and to demonstrate the functioning of the entities in a network simulated in the GNS3 software.

It is expected that this work can serve as an implementation base for a more robust OpenWRT device management solution.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS FONTE

Lista de abreviaturas

API	*Application Programming Interface*
SDN	*Software Defined Network*
DNS	*Domain Name System*
HTTP	*Hypertext Transfer Protocol*
IP	*Internet Protocol*
NAT	*Network Address Translation*
TCP	*Transport Control Protocol*
QoS	*Quality of Service*
DHCP	*Dynamic Host Configuration Protocol*
ICMP	*Internet Control Message Protocol*
WAN	*Wide Area Network*
LAN	*Local Area Network*
MD5	*Message Digest Algorithm*
RIP	*Routing Information Protocol*
VM	*Virtual Machine*
CPU	*Central Processing Unit*
MAC	*Media Access Control*
SQL	*Structured Query Language*
IoT	*Internet of Things*

Capítulo 1

Introdução

1.1 Contextualização

Redes de computadores sofreram inúmeras mudanças na última década. A adoção de paradigmas como computação em nuvem e redes definidas por software demonstram a necessidade de automação e centralização no gerenciamento de ambientes que estão se tornando cada vez maiores.

Ferramentas que facilitem o trabalho de administradores de redes e sistemas estão sendo amplamente empregadas para facilitar as operações dessas equipes que, muitas vezes, não possuem a quantidade de colaboradores necessários para manter um ambiente. Seja para sistemas operacionais, aplicações ou dispositivos de rede, a realização manual de configurações é algo que, para muitas empresas e organizações, não é mais plausível.

No caso de redes wireless, empresas e complexos industriais podem possuir centenas ou até mesmo milhares de dispositivos que proporcionem acesso à rede sem fio para seus colaboradores, sistemas e equipamentos. O OpenWRT (2.2) é uma distribuição GNU/Linux feita para sistemas embarcados, e é usado ao redor do mundo para proporcionar a seus usuários o total controle das funcionalidades de seus pontos de acesso. O uso desse tipo de sistema em ambientes de produção, nos quais é imprescindível a capacidade de aplicação de configurações necessárias em tempo hábil, se torna muito mais atrativo com a capacidade de um gerenciamento centralizado que replique através dos pontos de acesso gerenciados as mudanças configuradas.

1.2 Objetivos

1.2.1 Objetivo geral

Implementar uma arquitetura composta por uma controladora e um software que transforme sistemas OpenWRT em entidades gerenciáveis pela solução. A controladora, entidade que se assemelha a um controlador SDN, deve ser capaz de proporcionar uma experiência digna de gerenciamento remoto de pontos de acesso por um administrador autenticado.

1.2.2 Objetivos específicos

Os objetivos específicos tem como teor proporcionar uma visão mais detalhada do projeto. Desta forma, pretende-se:

- Confeccionar uma arquitetura de gerenciamento com entidades similares a uma rede SDN, cotendo uma controladora (OpenWRT-SDN-Controller) com uma interface de gerenciamento (NorthboundAPI) e interface de comunicação com dispositivos (SouthboundAPI). Além disso, a criação de um software que transforme dispositivos OpenWRT em entidades gerenciáveis pela controladora (OpenWRT-Daemon);
- Realizar a confecção de métodos na interface de gerenciamento que permitam a listagem de dispositivos, configurações e grupos lógicos, criação de grupo lógicos, segmentação de dispositivos em grupos lógicos e criação ou remoção de configurações nos dispositivos gerenciados;
- Realizar a confecção de métodos na interface de comunicação com dispositivos para permitir a autenticação, desautenticação e troca de chaves de autenticação de dispositivos gerenciados, além da criação de uma entidade para encaminhamento de configurações a dispositivos (Southbound Socket);
- Realizar a confecção de uma entidade interna da controladora (DB Daemon) para recebimento de requisições advindas de ambas as interfaces no intuito de realizar modificações no banco de dados SQL da solução;
- Realizar a confecção de um software para gerenciamento de dispositivos usando o sistema OpenWRT que contenha funcionalidades para criação de chave de autenticação, autenticação automática na controladora, método para recebimento de configurações e processamento das requisições recebidas (aplicação ou remoção);
- Realizar demonstrações do funcionamento dos métodos descritos da arquitetura implementada.

1.3 Justificativa

Apesar de ter sido lançado a muitos anos e possuir inúmeras funcionalidades para pontos de acesso [OpenWRT-Config 2016], o sistema OpenWRT não dispõe de compatibilidade nativa com sistemas externos que possibilitem o seu gerenciamento por uma controladora centralizada. A interface gráfica de configuração do sistema, denominada de LuCI, permite a criação de regras e configurações por uma página web. No entanto, o uso dessa solução em um ambiente com dezenas ou até centenas de dispositivos obrigaria administradores a entrar individualmente em cada uma deles para realizar suas operações. A proporção que redes, principalmente empresariais, estão tomando nos últimos anos [Wenfeng Xia 2015] torna esse tipo de gerenciamento ineficaz para a manutenção do bom funcionamento de uma rede.

1.4 Metodologia

Inicialmente, foi realizada uma pesquisa por possíveis trabalhos relacionados ou sistemas que se propuseram a realizar algo similar. Após os resultados dessa pesquisa, e motivado pelo o que a arquitetura se propõe a fazer, foi feita uma análise de controladoras SDN que poderiam ser adaptadas para realizar requisições nos formatos específicos que seriam abordados na solução. No fim, foi decidido que o melhor curso de ação seria a implementação de uma controladora (OpenWRT-SDN-Controller) feita especificamente para gerenciar dispositivos OpenWRT executando um software para integração com a entidade central (OpenWRT-Daemon).

Após a decisão de implementar a controladora e software de integração do zero, se tornou necessária a definição de um escopo de funcionalidades gerais da arquitetura e a concepção de entidades internas que permitissem a realização dessas capacidades. Diante da concepção das entidades, se iniciou o processo de implementação a partir das estruturas que estariam mais próximas ao administrador, uma abordagem top-down da arquitetura. Dessa forma, a confecção da solução se inciou da interface de gerenciamento (NorthboundAPI) e seus métodos, para em seguida montar o orquestrador do banco de dados (DB Daemon) responsável por receber as requisições e realizar modificações no banco de dados e, para finalizar a controladora, a interface de comunicação com os dispositivos gerenciados (SouthboundAPI). Esse fluxo de programação de entidades foi essencial para o desenvolvimento da controladora de forma intuitiva, uma vez que as funcionalidades eram implementadas a partir do seu recebimento na interface de gerenciamento, envio para o orquestrador do banco e encaminhamento pela interface de comunicação com os dispositivos, o que representa um fluxo natural de uma solução de gerenciamento.

O software de integração com a controladora (OpenWRT-Daemon) foi confeccionado de maneira a tornar um ponto de acesso OpenWRT que o execute em um dispositivo capaz de

ser gerenciado pela controladora implementada.

Por fim, foram feitos testes para validação do correto funcionamento das funcionalidades, estruturas e métodos planejados e descritos neste trabalho.

Capítulo 2

Fundamentação Teórica

Este capítulo tem como objetivo descrever de maneira introdutória conceitos essenciais para o entendimento da proposta da solução, seu objeto de gerenciamento e tecnologias empregadas para seu funcionamento. Além disso, serão expostos os trabalhos relacionados que se assemelham a este projeto, seja pela sua proposta ou tecnologias empregadas.

2.1 Software Defined Network

Nos últimos anos, o cenário de tecnologias em rede mudou drasticamente. A ampla adoção de automação nos processos operacionais em empresas, juntamente com implementação de infraestruturas em cloud para hospedagem de serviços, aponta para ambientes empresariais que prezam pela simplificação de ações tomadas pelos seus colaboradores.

Nessa linha, a arquitetura SDN (Software Defined Network) foi conceptualizada para facilitar o gerenciamento de dispositivos de rede por meio de uma controladora que centraliza todas as tomadas de decisão [Wenfeng Xia 2015]. No paradigma SDN, o controle do funcionamento de uma rede é desacoplado das funcionalidades de repasse de pacote de dispositivos e centralizado em scritps programáveis que definem sua operação [OpenNetworkFoundation 2013]. A inserção dessas mudanças em uma rede permite sua configuração dinâmica, programável e eficiente [Raphael Horvath 2015].

Para alcançar esse objetivo, uma arquitetura SDN propõe uma dissociação nos equipamentos de rede entre as funcionalidades de repasse de pacotes (data plane) e os controles que definem seu roteamento (control plane)[Manish Paliwal 2018]. Dessa maneira, uma rede SDN seria composta por controladoras que, a partir de programas confeccionados pelos seus administradores, realizam a tomada de decisão do roteamento de pacotes (control plane) recebidos por dispositivos de rede gerenciados (data plane). Para permitir a criação de novas configurações e se comunicar com os equipamentos gerenciados, a controladora SDN possui duas interfaces: a interface norte (gerenciamento) e a interface sul (comunicação com dispositivos) [Nick Feamster 2013]. A interface norte pode possuir plataformas ou APIs para

permitir o recebimento de novas configurações advindas de um administrador, enquanto a interface sul se comunica com os dispositivos gerenciados enviando as ações a serem tomadas com o tráfego recebido. Para se comunicar com os dispositivos, o protocolo Openflow [OpenNetworkFoundation 2012] é o método de comunicação mais usado por controladoras SDN. A partir dele é possível o envio de diferentes tipos de comandos para os dispositivos com o objetivo de realizar ações com os pacotes de rede como repassar, bloquear, modificar, dentre outras.

2.2 OpenWRT

O OpenWRT [OpenWRT-About 2016] é um sistema operacional para dispositivos embarcados, tipicamente usado por pontos de acesso. O sistema foi desenvolvido do zero para possibilitar a configuração de diversas tecnologias em dispositivos inseridos em redes empresariais ou residenciais. Diferentemente de outros sistemas usados por fabricantes de equipamentos de rede, o OpenWRT [Fainelli 2008] não visa ser um firmware estático usado por pontos de acesso, mas sim um sistema operacional GNU/Linux que possibilite a um usuário a seleção de aplicações e configurações [OpenWRT-Config 2016] para melhor suprir suas necessidades.

2.3 SQLite3

O SQLite [SQLite-About] é uma biblioteca que implementa bancos de dados SQL em arquivos que não dependem de servidores e configurações prévias para estarem funcionais. Por sua simplicidade de uso, o SQLite é a biblioteca mais usada para instanciar bancos de dados no mundo [SQLite-Most-Used], fazendo parte do código fonte de projetos amplamente utilizados de empresas como Adobe, Airbus, Apple, dentre outras [SQLite-High-Profile].

2.4 Trabalhos relacionados

Visando acompanhar o progresso nas tecnologias de redes de computadores, outros projetos já se propuseram em mesclar características do paradigma SDN com dispositivos OpenWRT.

Em [Muxi Yan 2016] é proposta uma extensão do protocolo OpenFlow para possibilitar o gerenciamento em nível de pacotes presente no framework SDN em dispositivos wireless (OpenWRT). O artigo tem como objetivo a integração de protocolos wireless no paradigma SDN a partir de abstrações na interface dos dispositivos que possibilitem a execução de ações como transição de usuários entre pontos de acesso, otimização de conexão por pontos de acesso mais próximos, políticas de QoS, dentre outras. Apesar de abordar temáticas

similares as trabalhadas no projeto deste documento, o artigo foca no gerenciamento de dispositivos OpenWRT em nível de pacotes, algo que não é objetivo da solução proposta neste trabalho.

Em [Hi Châu Lê 2020] foi apresentada uma implementação de um protótipo de um ponto de acesso baseado em SDN para redes IoT. Usando um dispositivo Raspberry Pi 3 com o sistema OpenWRT, os pesquisadores usaram o software Open vSwitche para receber configurações de uma controladora POX [POX-Controller] e definir o processamento de pacotes feito no dispositivo. De maneira similar ao artigo anterior, essa proposta foca no gerenciamento de um ponto de acesso em nível de pacotes, algo que não é objetivo da solução proposta neste trabalho.

Assim como descrito nos objetivos (Seção 1.2) deste documento, a implementação da arquitetura proposta se baseia no paradigma SDN para proporcionar uma solução de automação de configurações de dispositivos OpenWRT de maneira centralizada. Diante desses objetivos, a pesquisa inicial em busca de trabalhos que já implementaram algo similar me fez encontrar o software OpenWISP [OpenWISP-Architecture].

Desenvolvido inicialmente em 2008 para um projeto do governo italiano [Maurizio Goretti 2012] com objetivo de criar uma rede wireless pública na província de Roma, o software é composto por uma série de entidades que tornam possível o gerenciamento granular de dispositivos OpenWRT que compõem uma rede. As principais entidades do OpenWISP são:

- **OpenWISP Users** - Entidade responsável por proporcionar o gerenciamento da arquitetura por meio de plataformas web e APIs REST;
- **OpenWISP Controller** - Entidade responsável pela criação e gerenciamento de templates para configurações dos dispositivos. Possibilita o provisionamento automático de túneis VPN, execuções de shell scripts, inicialização de conexões SSH, dentre outras ações;
- **OpenWISP Monitoring** - Entidade que realiza o monitoramento de métricas do sistema dos dispositivos OpenWRT gerenciados. Coleta informações como uso de recursos, número de usuários conectados, dentre outras;
- **OpenWISP Network Topology** - Entidade que possibilita a visualização da topologia gerenciada por meio de informações obtidas de protocolos mesh;
- **OpenWISP Firmware Upgrader** - Entidade que possibilita a atualização de firmware do sistema OpenWRT;
- **OpenWISP Radius** - Entidade que proporciona funcionalidades para ampliar os métodos de autenticação de um usuário;

- **OpenWISP Notifications** - Entidade responsável pelo envio de notificações web e e-mails para alertar administradores acerca de eventos críticos na rede;

O OpenWISP, diante da quantidade de funcionalidades e similaridades que possui com a concepção inicial deste projeto, foi a maior inspiração para o desenvolvimento da arquitetura implementada.

Capítulo 3

Arquitetura proposta

Neste capítulo será descrita de maneira geral a arquitetura conceptualizada para gerenciamento de dispositivos OpenWRT e as entidades que a compõe.

Seguindo o paradigma de redes definidas por software, a arquitetura implementada neste trabalho é composta por duas entidades principais:

- **OpenWRT-SDN-Controller** - Entidade centralizadora das funcionalidades de gerenciamento dos dispositivos OpenWRT autenticados na solução.
- **OpenWRT-Dameon** - Software responsável por tornar um dispositivo OpenWRT que o execute em uma entidade passível de gerenciamento pela controladora proposta.

Na Figura 3.1 pode-se observar a visualização da arquitetura e a maneira como as entidades se interagem. A partir de requisições enviadas por um administrador, a controladora processa e repassa todos os parâmetros recebidos e prossegue com a resposta da requisição ou envio de configurações para os dispositivos gerenciados.

3.1 OpenWRT-SDN-Controller

O OpenWRT-SDN-Controller é a entidade da arquitetura que se assemelha conceitualmente à uma controladora SDN convencional. Assim como no paradigma SDN, a controladora, confeccionada do zero para essa arquitetura, possui uma interface de gerenciamento e uma interface de comunicação com os dispositivos gerenciados.

Durante o processo de idealização dessa entidade, foi decidido que, para proporcionar uma experiência digna de gerenciamento dos dispositivos, a controladora terá funcionalidades de:

- Listagem de dispositivos, configurações e grupos lógicos;

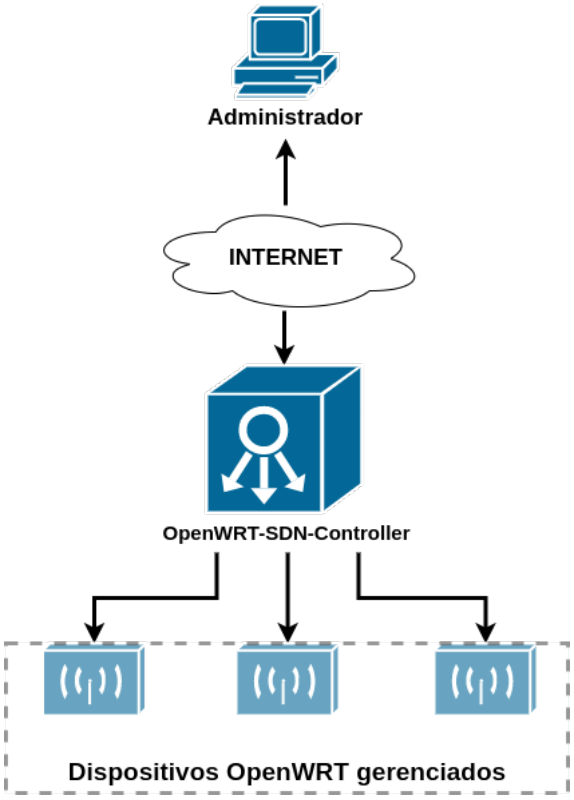

Figura 3.1: Arquitetura proposta.

- Criação de grupos lógicos;
- Segmentação de dispositivos em grupos lógicos;
- Sistema para recebimento, validação e envio de configurações para os dispositivos gerenciados.

Por mais que se assemelhe com uma controladora SDN convencional, a entidade criada neste trabalho foi feita especificamente para gerenciamento de dispositivos OpenWRT, e, para ser capaz de suprir as funcionalidades gerais descritas, é composta pelas entidades:

- **NorthboundAPI** - API HTTP para comunicação entre o administrador e a controladora. Entidade que possibilita ao administrador a visão e controle de todos os dispositivos autenticados na solução;
- **SouthboundAPI** - API HTTP responsável por toda comunicação entre a controladora e os dispositivos gerenciados;
- **DB Daemon** - Entidade responsável pelo processamento de requisições (advindas da NorthboundAPI e SouthboundAPI) para criação de queries apropriadas para modificações no banco de dados da solução.

A interação entre as entidades descritas acima pode ser observada na Figura 3.2. A seguir será detalhadamente detalhadamente o funcionamento de todos os métodos, funcionalidades

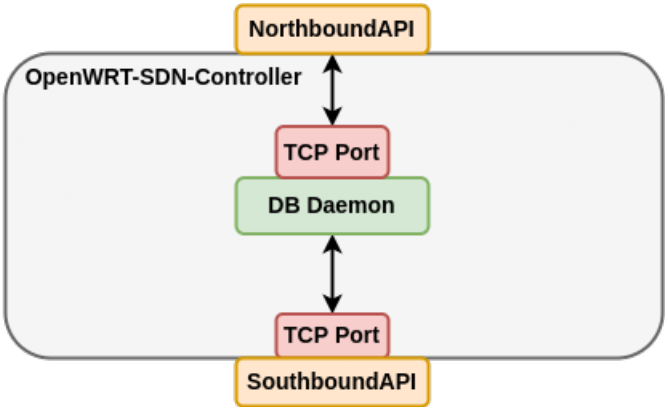

Figura 3.2: Interações entre as entidades que compõe o OpenWRT-SDN-Controller.

e estruturas existentes nas entidades que compõem o OpenWRT-SDN-Controller. Também serão descritas as funções das bibliotecas de utilitários *southutils.py* e *northutils.py*, juntamente com a exposição de fluxogramas de execução dessas entidades no intuito de auxiliar o entendimento de seu comportamento.

3.1.1 Configuração e inicialização

Para inicializar o OpenWRT-SDN-Controller é necessário executar o shell script *main.sh* existente no diretório do projeto. Esse script executa em paralelo os progamas em Python3 das entidades, sendo eles: *northboundAPI.py*, *southboundAPI.py* e *db_daemon.py*. O fluxograma de inicialização das entidades pode ser observado na Figura 3.3.

Durante a inicialização desses entidades, as funções *northutils.startup_north()*, *southutils.startup_south()* e *startup_db()* realizam uma busca no banco de dados SQLite3 da controladora para coletar informações como endereços, portas e token de autenticação. Por serem executadas como processos independentes, as entidades da controladora possuem funcionalidades de rede (APIs e sockets) associadas a endereços IP e portas diferentes. Por padrão as configurações de rede são:

- **NorthboundAPI** - IP: 127.0.0.1, Porta: 8080;
- **SouthboundAPI** - IP: 127.0.0.1, Porta: 8081;
- **SouthboundSocket** - IP: 127.0.0.1, Porta: 65002;
- **DB Daemon** - IP: 127.0.0.1, Porta: 65001;

Para modificar essas configurações no intuito de se adaptar a rede na qual está inserida, a controladora possui o script *config_cli.py* que funciona como uma CLI para definição de novos endereços e portas.

3.1.1.1 Criação de token para autenticação

Para validar o remetente de uma requisição recebida pela NorthboundAPI é usado um token de autenticação criado durante a primeira inicialização do OpenWRT-SDN-Controller. Esse token, gerado pela função *northutils.create_token()*, consiste de um hash MD5 de uma string de 150 caracteres pseudo-aleatórios. Essa informação é salva no banco de dados da controladora para ser utilizada por todos os métodos da API no estágio inicial de autenticação.

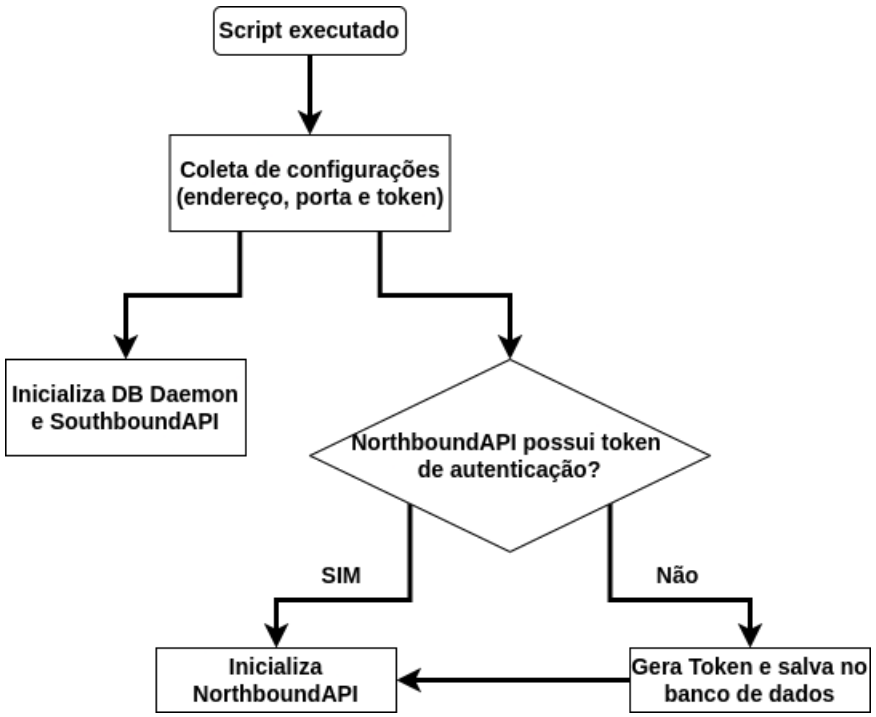

Figura 3.3: Fluxograma do processo de inicialização do OpenWRT-SDN-Controller.

3.1.2 NorthBoundAPI

Como descrita anteriormente, a NorthboundAPI é a interface que permite o uso das funcionalidades do OpenWRT-SDN-Controller. Por meio dela, um administrador autenticado na solução pode interagir com os métodos de listagem de configurações por parâmetros e por grupos, listagem e criação de grupos lógicos, agrupamento de dispositivos em grupos lógicos e criação ou remoção de configurações em dispositivos cadastrados.

3.1.2.1 Métodos suportados

Para suprir as funcionalidades descritas a controladora conta com os métodos:

- **/admin/config** - API HTTP POST para envio de configurações a serem aplicadas ou removidas dos dispositivos gerenciados;

- **/admin/host** - API HTTP POST para envio de requisição de segmentação de dispositivos cadastrados em grupos lógicos;
- **/admin/group** - API HTTP POST para envio de requisição de criação ou remoção de grupos lógicos;
- **/admin/list/group** - API HTTP GET para listagem de todos os grupos lógicos rastreados pela controladora;
- **/admin/list/host/<group_name>** - API HTTP GET para listagem de todos os hosts ou aplicação de filtro por nome de grupos lógicos;
- **/admin/list/config/dns/<params>** - API HTTP GET para listagem de todas as configurações Domain Name Server (DNS) que possuam os valores dos parâmetros passados;
- **/admin/list/config/qos/<params>** - API HTTP GET para listagem de todas as configurações Quality of Service (QoS) que possuam os valores dos parâmetros passados;
- **/admin/list/config/rip/<params>** - API HTTP GET para listagem de todas as configurações Routing Information Protocol (RIP) que possuam os valores dos parâmetros passados;
- **/admin/list/config/ipv4/<params>** - API HTTP GET para listagem de todas as configurações IPV4 que possuam os valores dos parâmetros passados;
- **/admin/list/config/dhcp/<params>** - API HTTP GET para listagem de todas as configurações Dynamic Host Configuration Protocol (DHCP) que possuam os valores dos parâmetros passados;
- **/admin/list/config/dhcp_relay/<params>** - API HTTP GET para listagem de todas as configurações DHCP Relay que possuam os valores dos parâmetros passados;
- **/admin/list/config/dhcp_relay/group/<group_name>** - API GTTP GET para listagem de todas as configurações DHCP Relay associadas a um grupo lógico;
- **/admin/list/config/dns/group/<group_name>** - API GTTP GET para listagem de todas as configurações DNS associadas a um grupo lógico;
- **/admin/list/config/rip/group/<group_name>** - API GTTP GET para listagem de todas as configurações RIP associadas a um grupo lógico;
- **/admin/list/config/qos/group/<group_name>** - API GTTP GET para listagem de todas as configurações QoS associadas a um grupo lógico;
- **/admin/list/config/ipv4/group/<group_name>** - API GTTP GET para listagem de todas as configurações IPV4 associadas a um grupo lógico;

- **/admin/list/config/dhcp/group/<group_name>** - API GTTP GET para listagem de todas as configurações DHCP associadas a um grupo lógico;

As funcionalidades internas de todos os métodos listados, juntamente com os parâmetros existentes para cada tipo de configuração suportada pela controladora, serão descritos em maiores detalhes nas próximas seções.

3.1.2.2 /admin/config

Esse é o método da NorthboundAPI responsável pelo recebimento de requisições para aplicação ou remoção de configurações nos dispositivos gerenciados. Esse método possui um formato específico no qual todas as configurações recebidas devem se encaixar. Pode-se observar o formato para requisições POST ao método */admin/config* na Listagem 3.1.

Listagem 3.1: Formato da requisição POST para /admin/config.

```
{
    "action":"value",
    "token":"value",
    "who":"value",
    "type":"value",
    "fields":{
        "name":"value",
        "name":"value"
    },
    "targets":{
        "group_name":["address","address"],
        "group_name":["address"]
    },
    "schedule":{
        "hour":"value",
        "minute":"value",
        "enable":"value",
        "dayofmonth":"value",
        "month":"value",
        "dayofweek":"value"
    }
}
```

O objetivo desse formato é ser o mais geral possível, de maneira a permitir a criação ou remoção de configurações de diferentes tipos, com diferentes parâmetros, para diferentes dispositivos. Os campos obrigatórios para uma configuração são:

- **action** - Define a ação a ser tomada nos dispositivos: *apply* para aplicar e *delete* para remover;
- **token** - Token de autenticação para aplicação da regra;

- **who** - Quem está aplicando a regra;
- **type** - Tipo da configuração (fw, dhcp, dhcp_relay, IPV4, RIP, QoS e DNS);
- **fields** - Dicionário cujas chaves são os nomes dos parâmetros refentes ao tipo da configuração e os valores as opções adotadas pelo remetente da requisição;
- **targets** - Dicionário cujas chaves são os nomes dos grupos lógicos aos quais os dispositivos alvos da configuração pertencem, e os valores os endereços IP associados ou a string "all"para todos os dispositivos do grupo;
- **schedule** - Dicionário cujas chaves são os parâmetros para agendamento de comandos na crontab. Campos usados para agendamento de configurações que possam interromper o serviço do dispositivo gerenciado.

Para facilitar a visualização, a Listagem 3.2 exemplifica uma configuração real que pode ser aplicada na controladora. Na imagem pode-se observar uma requisição para aplicação de uma regra de firewall para bloqueio de tráfego ICMP vindo do host *192.168.0.5*. Essa regra seria aplicada ao ponto de acesso *192.168.10.12* pertencente ao grupo *Default* no horário definido pelos valores do campo *schedule*.

Listagem 3.2: Exemplo da requisição POST para /admin/config.

```
{
    "action":"apply",
    "token":"349346313
        fde5f8437ec6358bc510164",
    "who":"admin",
    "type":"fw",
    "fields":{
        "name":"Reject LAN to WAN for custom
            IP",
        "src":"lan",
        "dest":"wan",
        "proto":"icmp",
        "target":"REJECT",
        "src_port":5000,
        "src_ip":"192.168.0.5"
    },
    "targets":{
        "Default":["192.168.10.12"]
    },
    "schedule":{
        "hour":"10",
        "minute":"1",
        "enable":"1",
        "dayofmonth":"10",
        "month":"11",
        "dayofweek":"12"
```

```
25          }
26  }
```

A seguir se encontram todas as configurações suportadas pela NorthbounAPI, juntamente com seus parâmetros, seu tipo, obrigatoriedade e descrição. Sendo elas:

- **firewall** - Configura o firewall de dispositivos OpenWRT autenticados na controladora. Descrito na Tabela 3.1;
- **DHCP** - Configura o servidor DHCP de dispositivos OpenWRT autenticados na controladora. Descrito na Tabela 3.2;
- **DHCP Relay** - Configura um redirecionamento de requisições DHCP para outro servidor. Descrito na Tabela 3.3;
- **IPV4** - Configura o endereço IPV4 de interfaces de dispositivos OpenWRT gerenciados. Descrito na Tabela 3.4;
- **RIP** - Configura rotas estáticas RIP em dispositivos OpenWRT. Descrito na Tabela 3.5;
- **QoS** - Configura regras de QoS em interfaces de dispositivos OpenWRT. Descrito na Tabela 3.6;
- **DNS** - Configura endereço DNS usado por dispositivos OpenWRT gerenciados. Descrito na Tabela 3.7.

Tabela 3.1: Parâmetros para configuração de firewall

Nome	**Tipo**	**Obrigatório**	**Descrição**
name	string	sim	Nome da regra
src	string	não	Zona lógica de origem (wan ou lan)
src_ip	string	não	Endereço IP de origem
src_mac	string	não	Endereço MAC de origem
src_port	integer	não	Porta de origem
proto	string	não	Tipo do protocolo
dest	string	não	Zona lógica de destino (wan ou lan)
dest_ip	string	não	Endereço IP de destino
dest_port	integer	não	Porta de destino
target	string	sim	Ação a ser tomada (ACCEPT, REJECT e DROP)

Para garantir que o formato especificado na Listagem 3.1 seja respeitado, o método */admin/config* conta com uma série de verificações. Esse processo é essencial para que a requisição recebida possa ser repassada com sucesso a todas as entidades internas do OpenWRT-SDN-Controller.

Tabela 3.2: Parâmetros para configuração de DHCP

Nome	**Tipo**	**Obrigatório**	**Descrição**
interface	string	sim	Interface que estará aguardando as requisições DHCP
start	integer	sim	Número que define o começo dos endereços oferecidos
limit_dhcp	integer	sim	Número que define o fim dos endereços ofertados
leasetime	string	sim	String do tipo: Número+h.

Tabela 3.3: Parâmetros para configuração de DHCP Relay

Nome	**Tipo**	**Obrigatório**	**Descrição**
id_relay	string	sim	Identificador da regra de relay
interface	string	sim	Interface que se conecta com o servidor DHCP alvo.
local_addr	string	sim	Endereço IP que escutará por requisições.
server_addr	string	sim	Endereço IP associado ao servidor DHCP alvo do relay.

O processamento de configurações se inicia com a validação do token de autenticação enviado no corpo da requisição. A validação é feita pela comparação com o token armazenado na base de dados da controladora, se forem iguais o processo continua, caso contrário a API retorna uma mensagem de erro.

Após a validação do token de autenticação é criado um objeto da classe *northutils.Config(corpo_req)* cujos métodos realizam uma série de validações no conteúdo enviado na requisição. O primeiro método a ser executado é o *Config.check_parameters_rule(parâmetros conhecidos)*, que recebe todos os parâmetros suportados para cada tipo de configuração no intuito de verificar se de fato a requisição recebida está de acordo com as definições. As verificações feitas pelo método da classe são:

- **(1)** - Conta a quantidade de campos recebidos e compara com a quantidade necessária (7);
- **(2)** - Verifica se o conteúdo do campo **type** está dentre as configurações suportadas;
- **(3)** - Verifica se os parâmetros enviados em **fields** estão de acordo com o campos obrigatórios para o tipo da configuração;
- **(4)** - Verifica se todos os parâmetros do campo **schedule** foram enviados.

Esse processo pode ser visualizado no fluxograma da Figura 3.8. Após a validação sintaxe da regra, se inicia a validação da existência dos grupos e entidades nas quais serão aplicadas as configurações. Para isso, inicialmente, são executadas três funções:

- *northutils.load_all_groups()* - Retorna um array com todos os grupos lógicos existentes no banco de dados;

Tabela 3.4: Parâmetros para configuração IPV4

Nome	Tipo	Obrigatório	Descrição
interface	string	sim	Interface na qual será aplicada a configuração.
ipaddr	string	sim	Endereço IPV4 a ser colocado.
netmask	string	sim	Máscara de rede do endereço.
gateway	string	sim	Endereço do gateway da rede.
dns	string	sim	Endereço do servidor DNS.

Tabela 3.5: Parâmetros para configuração RIP

Nome	Tipo	Obrigatório	Descrição
route_id	string	sim	Identificador da rota estática.
interface	string	sim	Interface a qual a rota será atribuída.
target	string	sim	Prefixo da rede destino (Ex: 192.168.0.4/24).
netmask	string	sim	Máscara da rede destino.
gateway	string	sim	Endereço assocaido ao gateway da rota.

- *northutils.load_host_group_relation(grupos_existentes)* - Recebe o array de grupos lógicos existentes e retorna um dicionário cujas chaves são os nomes dos grupos e o valor um array com todos os endereços pertencentes;
- *northutils.load_all_hosts()* - Retorna um dicionário com o endereço IP de todos os dispositivos autenticados na controladora;

Depois da coleta de todas essas informações, a API verifica se os nomes dos grupos passados no campo **targets** existem e se a configuração vai ser aplicada para todos os dispositivos do grupo lógico (**targets** = ["all"]), caso contrário, verifica se os endereços individuais passados estão autenticados e pertencem ao grupo lógico. Esse processo pode ser visualizado no fluxograma da Figura 3.9.

Após a validação dos parâmetros, grupos lógicos e hosts presentes na requisição, é realizado um processo essencial para identificação e rastreamento de regras na controladora, a criação de uma assinatura única para a configuração.

O processo de criação da assinatura é feito individualmente para cada endereço IP passado como argumento na requisição, de maneira a associar unicamente uma configuração aplicada a um dispositivo alvo. A função responsável pela criação da assinatura é a *northutils.hash_rule(requisição)* que recebe um dicionário contendo os campos: **action**, **token**, **who**, **type**, **fields**, **targets** (um único endereço IP) e **schedule**, que por sua vez são usados para gerar o identificador único. Para abordar todos os endereços enviados na configuração, o processo de criação da assinatura é feita em um loop iterativo que modifica o campo **targets** em cada repetição. Ao receber esse dicionário em cada iteração, a função cria uma série de strings que consistem da concatenação dos valores de cada campo. No caso do campo **fields** é realizada uma ordenação em ordem alfabética dos parâmetros recebidos para que a assinatura final seja a mesma independente da ordem na qual os parâmetros foram enviados.

Tabela 3.6: Parâmetros para configuração QoS

Nome	Tipo	Obrigatório	Descrição
interface	string	sim	Interface na qual a configuração será aplicada.
enabled	integer	sim	Define se a configuração está habilitada (0 ou 1).
classgroup	string	sim	Coloque "Default"por padrão.
overhead	integer	sim	Máscara da rede destino.
download	integer	sim	Define o limite do download em Kbits/s.
upload	integer	sim	Define o upload em Kbits/s.

Tabela 3.7: Parâmetros para configuração DNS

Nome	Tipo	Obrigatório	Descrição
address	string	sim	Endereço do servidor DNS a ser utilizado.

Por fim, é realizada a criação de um hash MD5 a partir de uma string final que é resultado da concatenação de todas as strings associadas aos parâmetros do dicionário. Esse hash é usado como assinatura que identifica unicamente uma configuração na controladora. Esse processo pode ser visualizado no fluxograma da Figura 3.10.

Após ser criada, a assinatura é inserida no corpo do dicionário da requisição recebida como o valor da chave *rule_hash*. Nesse momento se inicia a verificação da existência da configuração para evitar, no caso de criação, que uma configuração duplicada seja aplicada na controladora e nos dispositivos, e, no caso de remoção, que tente se remover uma configuração que não exista. Esse processo se inicia pela execução da função *northutils.load_all_applied_configs()*, que coleta todos os hashes de configurações existentes na controladora. A partir do resultado da função é feito um loop que verifica a existência de todos os hashes criados para a configuração recebida. Se a configuração não existir ela é inserida em um array de sucesso, caso contrário é inserida em um array de duplicatas, essas estruturas são usadas para, respectivamente, enviar configurações válidas a serem aplicadas no DB Daemon e retornar uma mensagem de erro informando para quais endereços a regra é repetida. Esse processo pode ser visualizado no fluxograma da Figura 3.11.

Para que uma configuração recebida seja cadastrada no banco de dados, é necessário o seu envio via protocolo TCP para o DB Daemon. Para que a entidade consiga processar os dados recebidos corretamente é adicionado uma chave (**global** = "config") no corpo da requisição antes dela ser enviada. Após a verificação da existência de todos os hashes gerados, a API */admin/config* finaliza seu processamento da requisição pelo envio do array de configurações não duplicadas ao DB Daemon, para isso é usado o método do *Config.send([array_nao_duplicadas])* relativo ao objeto instanciado no início do processamento da requisição. Esse método carrega as configurações do DB Daemon pela função *db_daemon.startup_db()* para coletar o IP e porta da entidade no intuito de iniciar uma comunicação TCP para envio iterativo das configurações. O processamento de configurações pelo DB Daemon será detalhado na Seção 3.1.4.

3.1.2.3 /admin/group

Esse é o método da NorthboundAPI responsável pelo recebimento de requsições para criação ou remoção de grupos lógicos na controladora. Assim como o */admin/config*, esse método possui um formato específico, no qual todas as requisições devem se encaixar. Pode-se observar o formato das requisições POST ao método */admin/group* na Listagem 3.3.

Listagem 3.3: Formato da requisição POST para /admin/group.

```
1 {
2       "action":"value",
3       "token":"value",
4       "who":"value",
5       "group_name":"value"
6
7 }
```

Os campos obrigatórios para uma requisição de criação ou remoção de grupos são:

- **action** - Define a ação a ser tomada pela controladora: *create* para criar e *delete* para remover;
- **token** - Token para autenticação da requisição;
- **who** - Quem está aplicando a regra;
- **group_name** Nome do grupo lógico a ser criado ou removido.

Para facilitar a visualização, a Listagem 3.4 exemplifica uma requisição real que pode ser enviada para o método de grupos lógicos da controladora. Na imagem pode-se observar uma requisição para criação do grupo lógico "Visitantes"no OpenWRT-SDN-Controller.

Listagem 3.4: Exemplo da requisição POST para /admin/group.

```
1 {
2       "action":"create",
3       "token":"349346313
            fde5f8437ec6358bc510164",
4       "who":"admin",
5       "group_name":"Visitantes"
6
7 }
```

Ao receber uma requisição para esse método, a NorthboundAPI inicialmente faz a sua autenticação pela validação do valor do campo **token**. Esse processo é feito pela comparação entre o valor recebido e o armazenado no banco de dados da solução.

Após a autenticação da requisição, é instanciado um objeto da classe *northutils.Config(requisição)* para em seguida usar o método *Config.check_group(grupos conhecidos)* no intuito de checar se a estrutura do dicionário recebido está de acordo com as definições descritas nessa seção. O argumento dessa função é um array com todos os grupos lógicos cadastrados na controladora, informação obtida através da execução da função *northutils.load_all_groups()*. A validação da requisição se inicia pela verificação da existência de todos os campos necessários e também checa se não foram enviados campos adicionais. Por fim, é validado se o grupo já existe, para não ser duplicado (**action** = create), ou para não tentar remover um grupo que não existe (**action** = delete). Esse processo pode ser observado no fluxograma da Figura 3.12.

Para cadastrar a mudança feita pela requisição validada no banco de dados, é necessário o seu envio via protocolo TCP para o DB Daemon. Para que a entidade consiga processar os dados recebidos corretamente é adicionado uma chave (**global** = "group") no corpo da requisição antes dela ser enviada. A função que realiza a comunicação TCP é o método *Config.send([group_config])* cujo argumento é um array de um único elemento, que no caso é a requisição recebida. O processamento feito pelo DB Daemon será detalhado na seção 3.1.4.

3.1.2.4 /admin/host

Esse é o método da NorthboundAPI responsável pelo recebimento de requisições para segmentação de hosts autenticados em grupos lógicos existentes na solução. Esse método possui um formato específico no qual todas as requisições devem se encaixar. Pode-se observar o formato das requisições POST ao método */admin/host* na Listagem 3.5.

Listagem 3.5: Formato da requisição POST para /admin/host.

```
{
    "action":"value",
    "token":"value",
    "who":"value",
    "group_name":"value",
    "targets":["address","address"]

}
```

Os campos obrigatórios do método são:

- **action** - Ação a ser tomada pela controladora: *insert* para inserir um dispositivo em um grupo, *update* para atualizar a qual grupo um dispositivo pertence e *delete* para remover um dispositivo de um grupo;
- **token** - Token de autenticação para a requisição;
- **who** - Quem está aplicando a regra;

- **group_name** - Nome do grupo lógico no qual os dispositivos serão inseridos ou removidos;
- **targets** - Array com os endereços de todos os dispositivos alvos da requisição;

Para facilitar a visualização, a Listagem 3.6 exemplifica uma requisição real que pode ser enviada para o método de segmentação de grupos lógicos da controladora. Na imagem, pode-se observar uma requisição para remoção dos dispositivos com endereço "192.168.15.17"e "192.168.22.129"do grupo lógico nomeado "Evento".

Listagem 3.6: Exemplo de requisição POST para /admin/host.

```
1 {
2       "action":"delete",
3       "token":"349346313
            fde5f8437ec6358bc510164",
4       "who":"admin",
5       "group_name":"Evento",
6       "targets
            ":["192.168.15.17","192.168.22.129"]
7
8 }
```

Após a autenticação da requisição pelo valor do campo **token**, é instanciado um objeto da classe *northutils.Config(requisição)* para em seguida usar o método *Config.check_host(hosts, grupos_conhecidos, relações_entre_hosts_e_grupos)*. Essa função recebe todos o endereço de todos os dispositivos cadastrados, todos os grupos lógicos existentes e um dicionário com a relação de pertencimento de dispositivos em grupos. O processo de validação se inicia com a verificação da existência de todos os campos obrigatórios para o método da API, em seguida válida se:

- **insert** e **update** - Grupo lógico passado em **group_name** existe e endereços em **targets** estão cadastrados na controladora;
- **delete** - Verifica se o grupo existe para em seguida validar se os dispositivos pertencem ao grupo, para que assim seja possível os remover.

Ao fim dessas verificações, a requisição é enviada ao DB Daemon com um novo campo **global** = host para que a entidade identifique seu tipo e faça os processamentos no banco de dados de acordo. Mais detalhes na seção 3.1.4.

3.1.2.5 Listagem de configurações

Nesta seção será detalhado o processo de listagem de configurações por filtragem de parâmetros e grupos lógicos. Como pode ser observado na Seção 3.1.2.1, os métodos para listagem de configurações seguem dois padrões:

- */admin/list/config/<tipo config>/<parâmetros para filtro>*
- */admin/list/<tipo config>/group/<nome do grupo>*

Os tipos de configurações podem ser observados na Tabelas 3.1, 3.2, 3.3, 3.4, 3.5, 3.6 e 3.7, juntamente com os respectivos parâmetros.

No caso de uma listagem por parâmetros, a ordem nos métodos da API é:

- **dhcp** - */dhcp/<interface>/<start>/<limit_dhcp>/<leasetime>*
- **dhcp_relay** - */dhcp_relay/<id_relay>/<interface>/<local_addr>/<server_addr>*
- **ipv4** - */ipv4/<ipaddr>/<netmask>/<gateway>/<dns>*
- **rip** - */rip/<interface>/<target>/<netmask>/<gateway>*
- **qos** - */qos/<interface>/<enabled>/<classgroup>/<overhead>/<download>/<upload>*
- **dns** - */dns/<address>*

Em todos esses métodos são validados o tipo dos parâmetros, verificando se um endereço de fato é um IPv4, uma interface é uma string, um limite de download é um inteiro e assim por diante. Caso um administrador queira usar só alguns parâmetros como filtro basta preencher o valor dos outros campos com "all", o que indica aos métodos que esse parâmetro não será levado em consideração na pesquisa feita pelo DB Daemon. Após a validação do tipo dos dados recebidos, os métodos enviam as requisições para o DB Daemon pela função *northutils.send_list_query_to_db(query_type, params, subtipo)*, que recebe **query_type** = config_list, dicionário **params** com os parâmetros de filtro e **subtipo** que representa o tipo da configuração a ser listada. Essa função monta um dicionário com o campo **global** = query_type, **params** e **sub_config** = subtipo, informações que usadas pelas funções de listagem do DB Daemon. Em seguida, é executada a função *db_daemon.startup_db()* para coletar as informações necessárias no intuito de estabelecer uma conexão TCP que envia os dados da requisição ao DB Daemon.

Após os processamentos descritos na Seção 3.1.4.6, a função recebe o resultado da listagem e o envia para que o método da NorthboundAPI o retorne ao usuário que realizou a requisição.

O processo de listagem de configurações por grupos segue uma lógica similar a da listagem por parâmetros, caso um grupo não seja passado como argumento na requisição são configurados as informações enviadas ao DB Daemon de maneira a selecionar todas as configurações de um tipo específico.

3.1.2.6 Listagem de dispositivos autenticados

O método HTTP GET */admin/list/host/<group_name>* da NorthboundAPI é o responsável por listar os dispositivos autenticados na controladora, filtrando ou não por um grupo lógico.

Ao receber uma requisição nesse método, a entidade define o valor das variáveis que serão enviadas ao DB Daemon para que ele consiga identificar e processar corretamente o tipo da requisição. A primeira variável configurada é a **query_type** = host_list. A segunda variável verifica se algum parâmetro de filtro foi passado no campo *group_name* da requisição, caso não tenha tenha, o valor é inserido dentro do dicionário **params** = {"**group_name**":"all"}, caso contrário é executada a função *northutils.load_all_groups()* para verificar se o grupo passado como argumento existe na controladora. Após a verificação, em caso afirmativo é definido **params** = {"**group_name**":"nome do grupo "}, em caso negativo é retornada uma mensagem de erro.

Em seguida, a requisição é enviada para o DB Daemon pela função *northutils.send_list_query_to_db(query_type, params)*, que inicialmente define um dicionário contendo o identificador da requisição **global** e os parâmetros de filtro, para em seguida executar a função *db_daemon.startup_db()* para coletar as informações necessárias no intuito de estabelecer uma conexão TCP que envia os dados da requisição ao DB Daemon.

Após os processamentos descritos na Seção 3.1.4.6, a função recebe o resultado da listagem e o envia para que o método da NorthboundAPI o retorne ao usuário que realizou a requisição.

3.1.2.7 Listagem de grupos lógicos

O método HTTP GET */admin/list/group* da NorthboundAPI é o responsável por listar os grupos lógicos existentes na controladora.

Ao receber uma requisição nesse método, a entidade define o valor das variáveis que serão enviadas ao DB Daemon para que ele consiga identificar e processar corretamente o tipo da requisição, sendo eles: **query_type** = "group_list" e os parâmetros de filtro **params** = {"**group_name**":"all"}.

Em seguida, a requisição é enviada para o DB Daemon pela função *northutils.send_list_query_to_db(query_type, params)*, que inicialmente define um dicionário contendo o identificador da requisição **global** e os parâmetros de filtro, para em seguida executar a função *db_daemon.startup_db()* para coletar as informações necessárias no intuito de estabelecer uma conexão TCP que envia os dados da requisição ao DB Daemon.

Após os processamentos descritos na Seção 3.1.4.8, a função recebe o resultado da listagem e o envia para que o método da NorthboundAPI o retorne ao usuário que realizou a requisição.

3.1.3 SouthbounAPI

Como descrita anteriormente, a SouthboundAPI é a interface que se comunica com os dispositivos OpenWRT gerenciados pela OpenWRT-SDN-Controller. Para isso ela possui uma API com métodos que possibilitam a autenticação, desautenticação e troca de chaves, como também funcionalidades para envio de configurações aos dispositivos.

Uma visão interna do funcionamento da SouthboundAPI pode ser observado na Figura 3.4. A imagem contém três fluxos de comunicação, sendo eles:

- **(1)** - Envio de configurações para o DB Dameon pelo método /admin/config da NorthboundAPI.
- **(2)** - Após o processamento das configurações no banco de dados pelo DB Daemon, elas são enviadas para o *Southbound Socket*, que por sua vez as encaminha para os dispositivos gerenciados. Mais detalhes sobre como o DB Daemon realiza essa processamento na Seção 3.1.4.
- **(3)** - Recebimento de requisições para autenticação, desautenticação e troca de chaves vindas dos dispositivos gerenciados.

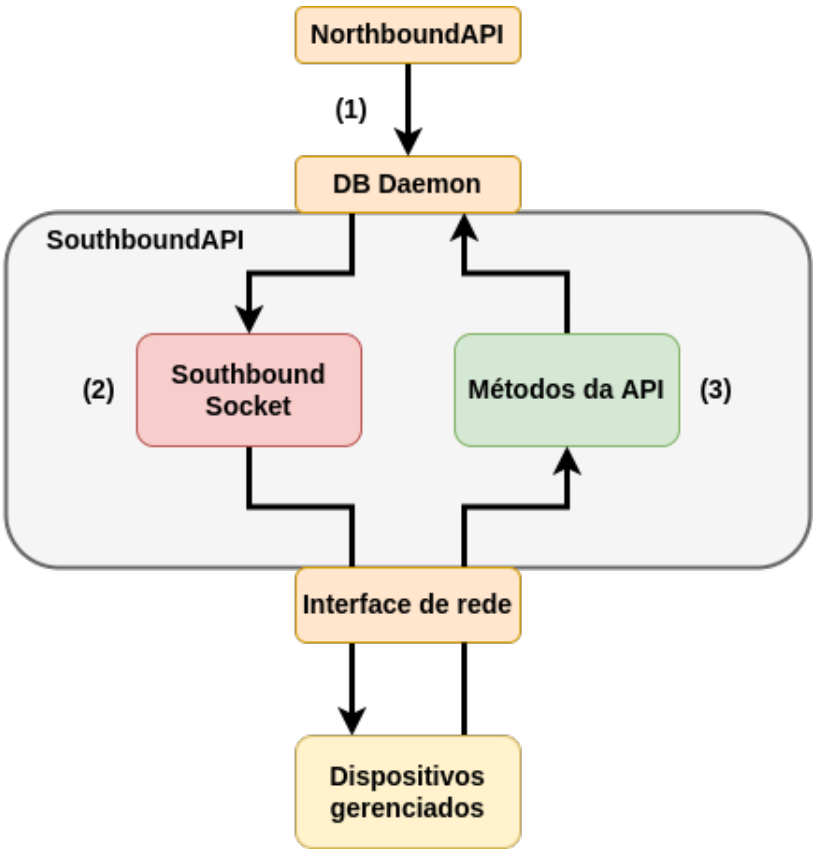

Figura 3.4: Entidades internas da SouthboundAPI.

Mais detalhes sobre como os dispositivos gerenciados executando o OpenWRT-Daemon enviam requisições para os métodos da SouthboundAPI estão presentes no Capítulo 3.2.

3.1.3.1 Métodos da southboundAPI

Para suprir essa funcionalidades a southboundAPI conta com os métodos:

- **/auth** - API HTTP POST que recebe requisições de autenticação de dispositivos OpenWRT executando o software OpenWRT-Daemon;
- **/deauth** - API HTTP POST que recebe requisições para desautenticação de dispositivos previamente autenticados;
- **/switch** - API HTTP POST que recebe requisições para troca de chaves de autenticação, usadas pelo OpenWRT-Daemon para validação de regras.

As funcionalidades internas dos métodos listados, juntamente com os parâmetros existentes para cada tipo de configuração suportada pela controladora, serão descritos em maiores detalhes nas próximas seções.

3.1.3.2 /auth

É o método da SouthboundAPI responsável pelo recebimento e processamento de requisições de dispositivos OpenWRT que buscam ser gerenciados pela controladora presente em seu ambiente. Esse método possui um formato específico para suas requisições que pode ser observado na Listagem 3.7.

Listagem 3.7: Formato da requisição POST para o método /auth.

```
1 {
2       "address":"value",
3       "port":"value",
4       "netmask":"value",
5       "token":"value"
6 }
```

Os campos obrigatórios são:

- **addres** - Endereço do dispositivo OpenWRT;
- **port** - Porta TCP na qual estará a API associada ao recebimento e processamento de configurações;
- **netmask** - Máscara de rede do endereço;
- **token** - Token para autenticação no OpenWRT-Daemon, usado pela controladora no envio das regras;

Para facilitar a visualização, a Listagem 3.8 exemplifica uma requisição real que é enviada de um dispositivo executando o OpenWRT-Daemon no intuito de se autenticar a uma controladora. Na imagem pode-se observar uma requisição de um dispositivo que tem endereço "192.168.15.2"com máscara de rede "255.255.255.0"e que está executando uma API para recebimento e processamento de configurações autenticadas pelo campo **token** na porta 50000.

Listagem 3.8: Exemplo de requisição POST para o método /auth.

```
1 {
2       "address":"192.168.15.2",
3       "port":"50000",
4       "netmask":"255.255.255.0",
5       "token":"1
            cb5986a34d4ab38741839a326f87ab5"
6 }
```

Antes de enviar os dados de uma requisição de autenticação recebida para o DB Daemon, a SouthboundAPI faz uma série de validações acerca da informações. Primeiramente, é validado se a estrutura da requisição está de acordo com a definição exposta anteriormente. Para isso é executada a função *southutils.check_auth_params(requisição)* que verifica se o valor do campo **address** é um endereço IPV4 válido, se o endereço já está cadastrado na controladora (evitar duplicatas), e por fim se o valor do campo **port** é um inteiro. Em seguida é executada a função *southutils.get_known_tokens()* para validar se o token de autenticação do dispositivo gerenciado já existe. Após essas verificações, é iniciada uma comunicação TCP com o DB Daemon para o envio da requisição com um novo campo **global** = auth, inserido para que a entidade de processamento de queries no banco de dados saiba identificar o tipo da requisição. Mais detalhes sobre o processamento feito pelo DB Daemon na Seção 3.1.4.9.

3.1.3.3 /deauth

Esse é o método da SouthboundAPI responsável pelo recebimento e processamento de requisições para desautenticação vindas de dispositivos previamente cadastrados na controladora. O formato da requisição pode ser observado na Listagem 3.9.

Listagem 3.9: Estrutura de requisição para o método /deauth.

```
1 {
2       "token":"value",
3       "address":"value"
4
5 }
```

Os campos obrigatórios são:

- **token** - Token único do dispositivo para autenticação da requisição;

- **address** - Endereço do dispositivo gerenciado.

Para facilitar a visualização, a Listagem 3.10 exemplifica uma requisição real que é enviada de um dispositivo OpenWRT-Daemon no intuito de se autenticar a uma controladora. Na imagem pode-se observar uma requisição para desautenticação do dispositivo com endereço "192.168.15.2".

Listagem 3.10: Exemplo de requisição para o método /deauth.

```
1 {
2       "token":"1
            cb5986a34d4ab38741839a326f87ab5",
3       "address":"192.168.15.2"
4
5 }
```

Antes de enviar os dados de uma requisição de desautenticação para o DB Daemon, a SouthboundAPI faz uma série de validações acerca das informações presentes no corpo. Primeiramente, é validado se a estrutura da requisição está de acorco com a definição exposta anteriormente. Para isso é executada a função *southutils.check_deauth_params(requisição)* que verifica a existência do campo **token**, o número de parâmetros enviados e se o valor de **address** é um endereço válido. Em seguida a API executa a função *southutils.get_known_tokens()* para verificar se o token recebido existe no banco de dados da controladora. Após essas verificações, é iniciada uma comunicação TCP com o DB Daemon para envio da requisição com um novo campo **global** = deauth, inserido para que a entidade de processamento de queries no banco de dados saiba identificar o tipo da requisição. Mais detalhes sobre o processamento feito pelo DB Daemon na Seção 3.1.4.10.

3.1.3.4 /switch

Esse é o método da SouthboundAPI responsável pelo recebimento e processamento de requisições para troca de chave de autenticação vindas de dispositivos previamente cadastrados na controladora. O formato da requisição pode ser observado na Listagem 3.11.

Listagem 3.11: Estrutura de requisição para o método /switch.

```
1 {
2       "token":"value",
3       "new_token":value"
4
5 }
```

Para facilitar a visualização, a Listagem 3.12 exemplifica uma requisição real que é enviada de um dispositivo OpenWRT-Daemmon no intuito de trocar a sua chave de autenticação. Na imagem pode se observar uma requisição de troca para **new_token** = *aij6hua34d4aba9k4m1l239a326f8ehj4*.

Listagem 3.12: Exemplo de requisição para o método /switch.

```
1 {
2       "token":"1
            cb5986a34d4ab38741839a326f87ab5",
3       "new_token":
            aij6hua34d4aba9k4m1l239a326f8ehj4"
4
5 }
```

Antes de enviar os dados de uma requisição de troca de chaves para o DB Daemon, a SouthboundAPI faz uma série de validações acerca das informações. Primeiramente, é validado se a estrutura da requisição está de acordo com a definição exposta anteriormente. Para isso é executada a função *southutils.check_switch_params(requisição)* que verifica se o campo **token** e **new_token** estão no corpo da requisição. Em seguida é executada a função *southutils.get_known_tokens()* para verificar se o valor de **token** existe no banco de dados da controladora e se o valor de **new_token** não existe. Após essas verificações, é iniciada uma comunicação TCP com o DB Daemon para o envio da requisição com um novo campo **global** = switch, inserido para que a entidade de processamento de queries no banco de dados saiba identificar o tipo da requisição. Mais detalhes sobre o processamento feito pelo DB Daemon na Seção 3.1.4.11.

3.1.3.5 Envio de configurações

O envio de configurações pela SouthboundAPI é feita por uma de suas entidades internas denominada *Southbound Socket*, como pode ser observado na Figura 3.4. Essa entidade, executada em paralelo com o servidor web que hospeda os métodos da API, é inicializada pela função *db_recv(socket_conifg)*, que por sua vez recebe as configurações de endereço e porta da entidade. A função realiza uma conexão HTTP POST nos endereços de dispositivos gerenciados, alvos da inserção ou remoção de uma regra, enviado um objeto contendo os parâmetros e valores de uma configuração. Mais detalhes sobre como ocorre o processamento de uma configuração nos dispositivos gerenciados podem ser encontrados no Capítulo 3.2.

3.1.4 DB Daemon

Como descrita anteriormente, o DB Daemon é a entidade responsável pelo processamento de requisições advindas da NorthboundAPI e SouthboundAPI para criação de queries SQL no intuito de modificar o banco de dados da controladora. Como pode ser observado nas seções anteriores, o DB Daemon é uma entidade essencial para todos os métodos da controladora, os quais, após a validação, enviam as requisições recebidas para a entidade com o objetivo de coletar ou inserir informações na base de dados.

Nesta seção será descrita a estrutura do banco de dados existente na controladora, o

processo de inicialização da entidade, a criação de queries SQL, e, no caso de uma requisição do */admin/config*, o envio de configurações para *Southbound Socket*.

3.1.4.1 Estrutura do banco de dados

O banco de dados da controladora consiste em uma série de arquivos do SQLite3 que contêm diferentes tabelas para armazenamento de diferentes dados relevantes a OpenWRT-SDN-Controller. Os arquivos que compõe o banco de dados são:

- **hosts_groups.db** - Arquivo do banco de dados que armazena os registros de dispositivos autenticados e grupo lógicos existentes. Possui as tabelas: *openwrt* e *groups*;
- **configs.db** - Arquivo do banco de dados que armazena todas as configurações criadas por um administrador. Possui as tabelas: *ipv4*, *dhcp*, *dhcp_relay*, *RIP*, *QoS*, *DNS* e *fw*;
- **configs_parameters.db** - Arquivo do banco de dados que armazena todos os tipos de configurações e seus parâmetros válidos, informações amplamente utilizadas no processo de verificação dos métodos da NorthboundAPI. Possui as tabelas: *ipv4*, *dhcp*, *dhcp_relay*, *RIP*, *QoS*, *DNS* e *fw*;
- **controller.db** - Arquivo do banco de dados que armazena as configurações de inicialização (endereço e porta) das entidades do OpenWRT-SDN-Controller. Possui as tabelas: *northboundAPI*, *southboundAPI*, *southbound_socket* e *db*.

Em uma nova instalação, o OpenWRT-SDN-Controller já vem com o banco de dados previamente construído para que seja possível a coleta e inserção de informações nele, porém, caso algum problema ocorra e esses arquivos sejam perdidos, o programa *generate_db.py* realiza a criação do zero de todos os arquivos e as tabelas que os compõe.

3.1.4.2 Inicialização e identificação de requisições

Ao ser executado, o programa do DB Daemon inicializa um socket TCP em um endereço e porta previamente configurados pelo administrador da solução. Esse socket aguarda por requisições vindas das outras entidades que compõem a controladora (NorthboundAPI e SouthboundAPI). Para identificar o tipo de processamento que deve fazer, a entidade verifica o campo **global** de todas as requisições. Os valores suportados para esse campo são:

- **config** - Identifica requisições de inserção ou remoção de configurações dos dispositivos gerenciados;
- **group** - Identifica requisições para criação ou remoção de grupos lógicos;
- **host** - Identifica requisições para segmentação de dispositivos gerenciados nos grupos lógicos existentes;

- **config_list** - Identifica uma requisição para listagem de configurações. Possui diferentes subtipos;
- **host_list** - Identifica uma requisição para listagem de dispositivos gerenciados pela controladora;
- **group_list** - Identifica uma requisição para listagem de grupos lógicos existentes;
- **auth** - Identifica uma requisição para autenticação vinda de um dispositivo executando o OpenWRT-Daemon;
- **deauth** - Identifica uma requisição para desautenticação vinda de um dispositivo executando o OpenWRT-Daemon;
- **switch** - Identifica uma requisição para troca de chaves de autenticação de um dispositivo gerenciado.

Nas próximas seções serão detalhados os processos de criação e execução de queries SQL para os tipos descritos.

3.1.4.3 Queries de configuração

Após o processamento de requisições para o método */admin/config*, descrito na Seção 3.1.2.2, o DB Daemon recebe a requisição para realizar as operações necessárias no banco de dados.

Ao identificar o tipo da requisição pelo campo **global** = config, é iniciada a função *process_recv_configs(requisição)*. Essa função inicialmente inicia uma conexão com o banco de dados *configs.db* para em seguida verificar qual o valor do campo **action** (apply ou delete). No caso de uma configuração nova a ser aplicada nos dispositivos (**action** = apply), é executada a função *create_query_config()* para criação de uma query SQL para inserção de dados. Essa função cria query a partir de duas strings:

- **(1)** - "insert into <tabela_da_config> (<campos_da_config>) ";
- **(2)** - "values (<valores_dos_campos>);"

Durante a execução da função, o nome da tabela (nome da configuração) e os nomes dos parâmetros da regra são inseridos para criar a primeira string. A segunda string é composta pelos respectivos valores dos parâmetros da primeira string. Após a criação das duas strings elas são concatenadas para criar a query final, que por sua vez será executada no banco de dados pela função *apply(query)*.

No caso do recebimento de uma requisição para remoção de uma configuração (**action** = delete), o DB Daemon executa a função *delete(requisição)*, que simplesmente executa uma

query SQL para remoção da configuração a partir de seu identificador único, sua assinatura hash.

Por fim, após a inserção ou remoção de uma configuração da base de dados, o DB Daemon encaminha as requisições para a SouthboundAPI no intuito de que elas as replique aos dispositivos presentes no campo **targets**, essa ação é feita pela função *db_send_to_south(requisição)*. A função usa os endereços dos dispositivos para coletar o valor de suas portas TCP e token, informações cadastradas no banco de dados da controladora durante o processo de autenticação. Após a obtenção desses dados, a função cria um dicionário contendo os campos obrigatório para envio ao OpenWRT-Daemon, sendo eles:

- **action** - Ação tomada pela regra (apply ou delete);
- **type** - Tipo da configuração;
- **fields** - Parâmetros das regras e seus valores;
- **port** - Porta TCP na qual o OpenWRT-Daemon está sendo executada no dispositivo gerenciado;
- **schedule** - Campos associados a crontab;
- **targets** - Endereço IP do dispositivo alvo da configuração;
- **rule_hash** - Assinatura hash que identifica unicamente uma configuração;
- **token** - Token usado para autenticação da requisição pelo OpenWRT-Daemon;

Mais informações sobre como o OpenWRT-Daemon processa requisições de configuração estão presentes na Seção 3.2.4.

3.1.4.4 Queries de grupos lógicos

Após o processamento de requisições para o método */admin/group*, o DB Daemon recebe a requisição para realizar as operações necessárias no banco de dados.

Ao identificar o tipo da requisição pelo campo **global** = group, é iniciada a função *process_recv_groups(requisição)*. Essa função inicialmente se conecta com o banco de dados *hosts_groups.db* para em seguida verificar qual o valor do campo **action** (create ou delete). No caso de uma requisição para criação de um novo grupo lógico (**action** = create), é criada uma query inserindo as informações do grupo lógico (group_name) na tabela *groups*. No caso do recebimento de uma requisição para remoção de um grupo (**action** = delete), é criada uma query que remove o grupo lógico a partir de seu nome e outra query para atualização do registro de todos os dispositivos autenticados que pertenciam ao grupo, os colocando no grupo "Default".

3.1.4.5 Queries de segmentação de dispositivos

Após o processamento de requisições para o método */admin/host*, o DB Daemon recebe a requisição para realizar as operações necessárias no banco de dados.

Ao identificar o tipo da requisição pelo campo **global** = host, é iniciada a função *process_recv_hosts(requisição)*. Essa função inicialmente se conecta com o banco de dados *hosts_groups.db* para em seguida verificar qual o valor do campo **action** (insert, update ou delete). No caso de uma requisição para inserção de um dispositivo em um grupo ou atualização do grupo ao qual pertence (**action** = insert ou **action** = update), a função inicia um loop para criação de queries para atualizar o campo *group_name* no banco de dados de todos os endereços presentes em **targets**.

Por outro lado, quando uma requisição possui o campo **action** = delete é iniciado um loop que atualiza o registro do banco de dados de todos os dispositivos no campo **targets** aos inserir no campo "Default". Além disso, são atualizados todos os registros de configurações associadas ao endereço de um dispositivo que está sendo removido de um grupo lógico para que elas reflitam as mudanças.

3.1.4.6 Queries de listagem de configuração

Após o processamento de requisições para os métodos descritos nas Seções 3.1.2.5, o DB Daemon recebe as requisições para realizar as operações necessárias no banco de dados.

Ao identificar o tipo da requisição pelo campo **global** = config_list, é realizada uma segunda verificação para identificar o seu subtipo (**sub_config**), que define qual o tipo de configuração que será listada. Ao identificar o subtipo da configuração, é executada a função *pre_send_list_result(requisição, query inicial, objeto da conexão com NorthboundAPI, subtipo)*, que por sua vez inicia a função *process_recv_config_list(requisição, query inicial)*.

Essa função pega os parâmetros (**params**) de listagem recebidos como argumento para realizar uma busca no banco de dados. Com esse objetivo, a função verifica se os parâmetros usados como filtro possuem valores específicos ou se referenciam todos os valores possíves (**params["nome do parâmetro"]** = all). Em seguida é iniciado um loop para criação de uma query que seleciona os registros existentes na tabela específica da configuração no banco do arquivo *configs.db*. Após a obtenção dos resultados, a função *pre_send_list_result(argumentos)* os insere em dicionários estruturados para cada tipo de listagem.

Por fim, um array com todos os registros que se encaixavam nos filtros da requisição são enviados de volta a NorthboundAPI pelo objeto da conexão passado anteriormente como argumento.

3.1.4.7 Queries de listagem de dispositivos

Após o processamento de requisições para o método */admin/list/host/<group_name>*, o DB Daemon as recebe para realizar as operações necessárias no banco de dados.

Ao identificar o tipo da requisição pelo campo **global** = host_list, é iniciada a função *process_recv_hosts_list(requisição)*. Essa função inicialmente se conecta com o banco de dados *hosts_groups.db* para em seguida verificar se foi passado algum nome de grupo lógico como filtro da listagem dos dispositivos gerenciados. Em caso afirmativo, a função seleciona todos os dispositivos autenticados que pertencem ao grupo lógico requisitado. Caso contrário, são selecionados todos os dispositivos autenticados na controladora.

O resultado da busca no banco de dados é retornado pela função *send_list_loop(resultado, objeto da conexão com a NorthboundAPI)*, que envia individualmente para a NorthboundAPI todos os registros presentes no array passado como argumento.

3.1.4.8 Queries de listagem de grupos lógicos

Após o processamento de requisições para o método */admin/list/group*, o DB Daemon as recebe para realizar as operações necessárias no banco de dados.

Ao identificar o tipo da requisição pelo campo **global** = group_list, é iniciada a função *process_recv_groups_list(requisição)*. Essa função inicialmente se conecta com o banco de dados *host_groups.db* para em seguida selecionar todos os grupos existentes na controladora.

O resultado da busca no banco de dados é retornado pela função *send_list_loop(resultado, objeto da conexão com a NorthboundAPI)*, que envia individualmente para a NorthboundAPI todos os registros presentes no array passado como argumento.

3.1.4.9 Queries de autenticação de dispositivos

Após o processamento de requisições para o método */auth* da SouthboundAPI, o DB Daemon as recebe para realizar as operações necessárias no banco de dados.

Ao identificar o tipo da requisição pelo campo **global** = auth, é iniciada a função *process_recv_auth(requisição)*. Essa função inicialmente se conecta com o banco de dados *hosts_group.db* para em seguida inserir na tabela *openwrt* as informações de um dispositivo que está se autenticando.

3.1.4.10 Queries de desautenticação de dispositivos

Após o processamento de requisições para o método */deauth* da SouthboundAPI, o DB Daemon as recebe para realizar as operações necessárias no banco de dados.

Ao identificar o tipo da requisição pelo campo **global** = deauth, é iniciada a função *process_recv_deauth(requisição)*. Essa função inicialmente se conecta com o banco de dados *hosts_groups.db* para em seguida remover o registro do dispositivo a partir do valor do seu token de autenticação. Após essa ação, a função se conecta ao banco de dados *configs.db* para remover todos os registros de configurações associadas ao dispositivo que está se desautenticando da controladora.

3.1.4.11 Queries de troca de chave de autenticação de dispositivos

Após o processamento de requisições para o método */switch* da SouthboundAPI, o DB Daemon as recebe para realizar as operações necessárias no banco de dados.

Ao identificar o tipo da requisição pelo campo **global** = switch, é iniciada a função *process_recv_switch(requisição)*. Essa função inicialmente se conecta com o banco de dados *hosts_groups.db* para em seguida executar um comando SQL no intuito de modificar o registro de um dispositivo ao trocar sua antiga chave de autenticação (**token**) pela nova (**new_token**).

3.2 OpenWRT-Daemon

Como dito anteriormente, o sistema operacional OpenWRT não possui por padrão uma API que possibilite a configuração remota de seus serviços, muito menos a partir de uma controladora centralizada.

O OpenWRT-Daemon é um programa desenvolvido para execução no sistema de pontos de acesso no intuito de os tornar gerenciáveis por um administrador a partir de comandos enviados ao OpenWRT-SDN-Controller. Com esse objetivo, o programa, após sua inicialização, executa uma API que é capaz de receber configurações autenticadas para aplicação no sistema.

Esta seção tem como objetivo descrever detalhadamente o funcionamento dos métodos, funcionalidades e estruturas existentes no OpenWRT-Daemon executado por dispositivos gerenciados. Também serão descritas as funções da biblioteca de utilitários *daemon_utils.py* juntamente com a exposição de fluxogramas de execução no intuito de auxiliar o entendimento do comportamento da entidade.

3.2.1 Estrutura do banco de dados

O banco de dados do OpenWRT-Daemon consiste de uma série de arquivos do SQLite3 que contêm diferentes tabelas para armazenamento de diferentes dados relevantes a entidade. Os arquivos que compõe o banco de dados são:

- **host_config.db** - Arquivo do banco de dados que armazena as informações de inicialização do OpenWRT-Daemon;
- **controller_config.db** - Arquivo do banco de dados que armazena as informações necessárias para comunicação com o OpenWRT-SDN-Controller;
- **configs.db** - Arquivo do banco de dados que armazena todas as configurações criadas por um administrador. Possui as tabelas: *ipv4*, *dhcp*, *dhcp_relay*, *RIP*, *QoS*, *DNS* e *fw*.

Em uma nova instalação, o OpenWRT-Daemon já vem com o banco de dados previamente construído para que seja possível a coleta e inserção de informações nele, porém, caso algum problema ocorra e esses arquivos sejam perdidos, o programa *generate_daemon_db.py* realiza a criação do zero de todos os arquivos e as tabelas que os compõe;

3.2.2 Configuração e inicialização

Para inicializar o OpenWRT-Daemon é necessário executar o script Python nomeado de *openwrt_daemon.py*. Esse script carrega as configurações da entidade e inicia sua API. Durante sua inicialização, o OpenWRT-Daemon executa a função *daemon_utils.startup_process()* para coletar no banco de dados da entidade informações como endereço, porta, token de autenticação, dentre outras. Antes de iniciar sua API, o programa verifica o resultado da função de inicialização para validar se já existe um token de autenticação cadastrado no banco, caso não exista são executadas as funções *create_token.create()* e *create_token.update_token(token)*. Mais detalhes sobre o processo de criação de tokens na Seção 3.2.3. Em seguida, é verificado se a entidade já está autenticada em uma controladora, caso não esteja é iniciada a função *try_auth(configs de inicialização)* que carrega do banco de dados as informações da controladora pela função *daemon_utils.controller_config()* para enviar ao método */auth* da SouthboundAPI os dados necessários para autenticação. Após essas verificações, é inicializada uma API HTTP que contém um único método nomeado de */config*, responsável pelo recebimento e processamento de configurações. O processo de inicialização pode ser observado na Figura 3.13.

As informações de endereço e porta associados ao OpenWRT-Daemon e OpenWRT-SDN-Controller podem ser configuradas por um administrador a partir da execução do script *config_cli.py*, que atua como uma interface de linha de comando para configuração das informações de inicialização e conexão da entidade.

3.2.3 Criação do token

O token é um dado essencial para o OpenWRT-Daemon que atua na autenticação de requisições de configuração advindas de uma controladora. Assim como descrito na seção anterior, as funções usadas para criação de tokens estão presentes no script *create_token.py*. Esse script possui duas funções:

- *create()* - Cria o token usado para autenticação de requisições;
- *update_token(token)* - Recebe o token criado e atualiza a base de dados do OpenWRT-Daemon.

A função *create()* se inicia pela obtenção de 150 caracteres pseudo-aleatórios que são concatenados em uma string para criação de um hash MD5 que atua como token. Após esse processamento, a função *update_token(token)* pode ser usada para se conectar ao banco de dados no arquivo *host_config.db* no intuito de atualizar o registro das informações do OpenWRT-Daemon.

3.2.3.1 Troca de chave de autenticação

Ao iniciar o script *create_token.py*, é verificado se existe um token associado ao OpenWRT-Daemon para em seguida validar com o administrador se ele deseja criar um novo token para autenticação de configurações. Após responder afirmativamente, são executadas as funções *create()* e *update_token(token)* para modificar o registro das configurações da entidade na base de dados local. Em seguida é criado um dicionário contendo o token antigo (**token**) e o novo (**new_token**) que serão enviados via uma requisição HTTP POST ao método */switch* da SouthboundAPI. Mais detalhes sobre como é feito o processamento de uma requisição para troca de chaves na Seção 3.1.3.4.

3.2.4 Processamento de regras

Ao receber uma requisição no método */config* (mais detalhes na Seção 3.1.4.3), o OpenWRT-Daemon carrega as suas configurações ao executar a função *daemon_utils.startup_process()* no intuito de verificar se o token recebido na requisição é o mesmo que está armazenado no banco de dados da entidade. Após a validação do token é verificado se a requisição recebida deseja adicionar (**action** = apply) ou remover (**action** = delete) uma configuração do sistema e da base de dados.

3.2.4.1 Criação de configurações

No caso de uma requisição para criação de uma configuração, inicialmente é executada a função *daemon_utils.create_query_config(requisição)* para criação de uma query SQL para

inserção da configuração no banco de dados. Após a criação da query de inserção é verificado qual o tipo da configuração para executar uma das funções responsáveis.

Firewall: Para esse tipo de configuração a função *daemon_utils.apply_firewall_config(fields, hash da regra)* é executada para criação de regras de firewall no serviço do firewalld do sistema OpenWRT. A configuração é feita pela inserção dos parâmetros no arquivo /etc/config/firewall em um formato específico que pode ser observado na Figura . Para ser capaz de criar esse formato dinamicamente, a função possui um dicionário cujas chaves são todos os possíveis parâmetros de uma configuração de firewall e os valores as strings que devem ser inseridas no arquivo. Para facilitar o rastreio de uma configuração específica, é inserido o hash da configuração antes de seu corpo no arquivo do firewall. Após a inserção da configuração é realizada a reinicialização do serviço firewalld para aplicação efetiva. Um exemplo de configuração no arquivo pode ser observado na Figura 3.5(a).

DHCP: Para esse tipo de configuração a função *daemon_utils.dhcp_config(requisição)* é executada para criação de configurações DHCP a partir do uso da API do sistema Unified Configuration Interface (UCI) pela execução de comandos nativos. Após a execução dos comandos UCI que configuram o serviço a partir dos novos parâmetros, o odhcpd é reinicializado para aplicação efetiva. Um exemplo de configuração no arquivo pode ser observado na Figura 3.5(b).

```
config rule
        option  name            'Reject LAN to WAN for custom IP'
        option  src             'lan'
        option  src_ip          '192.168.1.2'
        option  src_mac         '00:11:22:33:44:55'
        option  src_port        '80'
        option  dest            'wan'
        option  dest_ip         '194.25.2.129'
        option  dest_port       '120'
        option  proto           'tcp'
        option  target          'REJECT'
```

((a)) Exemplo de configuração de firewall no arquivo /etc/config/firewall.

```
# uci -N show dhcp.@dhcp[0]
dhcp.@dhcp[0]=dhcp
dhcp.@dhcp[0].interface='lan'
dhcp.@dhcp[0].start='100'
dhcp.@dhcp[0].limit='150'
dhcp.@dhcp[0].leasetime='12h'
```

((b)) Exemplo de configuração de DHCP no arquivo /etc/config/dhcp.

Figura 3.5: Configurações de firewall e DHCP

DHCP Relay: para esse tipo de configuração a função *daemon_utils.dhcp_relay_config(requisição)* é executada para criação de configurações de redirecionamento de queries DHCP pelo dispositivo OpenWRT gerenciado. A configuração é feita pela inserção dos parâmetros no arquivo /etc/config/dhcp em um formato específico que pode ser observado na Figura . Para ser capaz de criar esse formato dinamicamente, a função possui um dicionário cujas chaves são todos os possíveis parâmetros de uma configuração do DHCP Relay e os valores as strings que devem ser inseridas no arquivo. Para facilitar o rastreio de uma configuração específica, é inserido o hash da configuração antes de seu corpo no arquivo do serviço dhcp. Após a inserção da configuração é realizada a reinicialização do odhcpd para aplicação efetiva. Um exemplo de configuração no arquivo pode ser observado na Figura 3.6(a).

IPV4: para esse tipo de configuração a função *daemon_utils.ipv4_config(requisição)* é executada para criação de configurações IPv4 em interfaces do dispositivo a partir do uso da API do sistema Unified Configuration Interface (UCI) pela execução de comandos nativos. Após a execução dos comandos UCI que configuram o serviço a partir dos novos parâmetros, o serviço network é reinicializado para aplicação efetiva.

RIP: para esse tipo de configuração a função *daemon_utils.RIP_config(requisição)* é executada para criação de rotas de roteamento estático RIP no sistema OpenWRT gerenciado. A configuração é feita pela inserção dos parâmetros no arquivo /etc/config/network em um formato específico que pode ser observado na Figura . Para ser capaz de criar esse formato dinamicamente, a função possui um dicionário cujas chaves são todos os possíveis parâmetros de uma configuração de RIP e os valores as strings que devem ser inseridas no arquivo. Para facilitar o rastreio de uma configuração específica, é inserido o hash da configuração antes de seu corpo no arquivo de configuração da rede. Após a inserção da configuração é realizada a reinicialização do serviço network para aplicação efetiva. Um exemplo de configuração no arquivo pode ser observado na Figura 3.6(b).

```
config relay 'id'
        option interface 'lan'
        option local_addr '1.1.1.1'
        option server_addr '2.2.2.2'
```

((a)) Exemplo de configuração de DHCP Relay no arquivo /etc/config/dhcp.

```
config route 'route_example_1'
        option interface 'lan'
        option target '172.16.123.0'
        option netmask '255.255.255.0'
        option gateway '172.16.123.100'
```

((b)) Exemplo de configuração de RIP no arquivo /etc/-config/network.

Figura 3.6: Configurações de DHCP Relay e RIP

QoS: para esse tipo de configuração a função *daemon_utils.qos_config(requisição)* é executada para criação de uma configuração de QoS no serviço qos do sistema OpenWRT. A configuração é feita pela inserção de parâmetros no arquivo /etc/config/qos em um formato específico que pode ser observado na Figura . Para ser capaz de criar esse formato dinamicamente, a função possui um dicionário cujas chaves são todos os possíveis parâmetros de uma configuração de QoS e os valores as strings que devem ser inseridas no arquivo. Para facilitar o rastreio de uma configuração específica, é inserido o hash da configuração antes de corpo no arquivo de QoS. Após a inserção da configuração é realizada a reinicialização do serviço qos para aplicação efetiva. Um exemplo de configuração no arquivo pode ser observado na Figura 3.7.

```
config interface dsl
        option enabled      1
        option classgroup   "Default"
        option overhead     1
        option upload       512
        option download     4096
```

Figura 3.7: Exemplo de configuração de QoS no arquivo /etc/config/qos.

DNS: para esse tipo de configuração a função *daemon_utils.dns_config(requisição)* é executada para criação de uma configuração de DNS no dispostivo OpenWRT. A configuração é feita pela inserção do endereço do servidor DNS no arquivo /etc/resolv.conf antecedido pelo hash da configuração.

3.2.4.2 Remoção de configurações

No caso de uma requisição para remoção de uma configuração, inicialmente é executada a função *daemon_utils.detele_config(hash, tipo da regra, requisição)* que se conecta ao banco de dados do arquivo *configs.db* no intuito de remover o registro da configuração do banco de dados do OpenWRT-Daemon.

Firewall: Para esse tipo de configuração, a função executada para remoção de configuração inicialmente coleta da base de dados o registro da regra usando o hash como filtro. Esses dados são obtidos para determinar quantos parâmetros, dentre os possíveis, foram enviados, informação que é essencial para remoção da configuração do arquivo /etc/config/firewall. Após obter o número de parâmetros, é executado o comando "*sed -e '/hash/,+(número de parâmetros)d' -i /etc/config/firewall*", que por sua vez identifica a linha que contém o hash da regra e remove ela mais as N (número de parâmetros) linhas seguintes. Por fim, o serviço firewall é reinicializado para aplicação das mudanças.

DNS: para esse tipo de configuração, a função executada para remoção da configuração simplesmente executa o comando "*sed -e '/hash/,+1d' -i /etc/resolv.conf*"para identificação da linha que contém o hash da configuração no intuito de a remover juntamente com a próxima linha (+1d), que contém o endereço do servidor DNS configurado.

IPV4: para esse tipo de configuração, a função executada faz o uso de comandos do UCI para deletar os valores de endereço, máscara de rede, gateway e DNS de uma interface previamente configurada. Por fim, o serviço network é reinicializado para aplicação das mudanças.

DHCP: para esse tipo de configuração, a função executada faz o uso de comandos do UCI para deletar os valores que definem a interface associado ao serviço DHCP, o início de endereços, o fim de endereços e tempo limite do empréstimo. Por fim, o serviço odhcpd é reinicializado para aplicação das mudanças.

DHCP Relay: para esse tipo de configuração, a função executada para remoção simplesmente executa o comando "*sed -e '/hash/,+4d' -i /etc/config/dhcp*" para identificar a linha que contém o hash da configuração no intuito de a remover juntamente com as quatros próxiams linhas (+4d), que contém os parâmetros obrigatórios da configuração. Por fim, o serviço odhcpd é reinicializado para aplicação das mudanlas.

QoS: para esse tipo de configuração, a função executada para remoção simplesmente executa o comando "*sed -e '/hash/,+6d' -i /etc/config/qos*" para identificar a linha que contém o hash da configuração no intuito de a remover juntamente com as seis próximas linhas

(+6d), que contém os parâmetros obrigatórios da configuração. Por fim, o serviço qos é reinicializado para aplicação das mudanças.

RIP: para esse tipo de configuração, a função executada para remoção simplesmente executa o comando "*sed -e '/hash/,+5d' -i /etc/config/network*" para identificar a linha que contém o hash da configuração no intuito de a remover juntamente com as cinco próximas linhas (+5d), que contém os parâmetros obrigatórios da configuração. Por fim, o serviço network é reinicializado para aplicação das mudanças.

3.2.4.3 Agendamento de configurações

O agendamento de uma configuração é feito pelo uso dos parâmetros presentes no campo **schedule** de uma configuração recebida. Para que o método */config* do OpenWRT-Daemon identifique que uma configuração deve ser aplicada imediatamente, é necessário definir o valor do campo **enabled** = 1. Caso contrário, o método vai processar a requisição de maneira a criar um registro na crontab do sistema para executar a configuração no momento definido.

Ao identificar que uma requisição deve ser agendada, o Openwrt-Daemon executa a função *daemon_utils.cron_create(rule)* que recebe os dados enviados pela controladora. Essa função realiza a criação de um script em Python cujo nome é *rule_hash .py*, sendo o valor de *rule_hash* a assinatura única da configuração recebida. Esse script possui o dicionário da requisição recebida, o dicionário de configurações do OpenWRT-Dameon e o path do sistema até o local onde o código foi criado. Além disso, a função insere no arquivo */etc/crontabs/root* o registro para execução do script criado antecedido pelo hash da configuração.

No momento definido pelos parâmetros inseridos na crontab, o script Python é executado. Para aplicar a regra diretamente, o script faz uma requisição HTTP POST para o endereço e porta do OpenWRT-Daemon no método */config*, para que a configuração armazenada no corpo de seu código seja de fato aplicada. Para diferenciar uma requisição padrão de uma requisição advinda de um processo do crontab, o dicionário da configuração recebe uma nova chave nomeada de **cron**. Após enviar a requisição para o método, com o objetivo de que ela seja aplicada ou removida do sistema, o script executa o comando "*sed -e '/hash/,+1d' -i /etc/crontabs/root*" para remover o registro do agendamento da crontab. Em seguida, o script Python se auto-remove do sistema uma vez que já cumpriu seu papel.

3.2.5 Desautenticação

O processo de desautenticação de um dispositivo gerenciado pela controladora é feito pela execução de um script nomeado *deauth.py* presente no diretório do projeto. Esse script possui uma única função *deauth()* que coleta as informações do OpenWRT-Daemon e da OpenWRT-SDN-Controller pelas funções *daemon_utils.startup_process()* e *daemon_utils.controller_config()*. Após a coleta das informações retornadas por essas funções, o programa envia uma requisição HTTP POST para o método */deauth* da Southboun-

dAPI contendo os campos **token** e **address**, que serão usados pelo DB Daemon para remover todos os registros associados a esse dispositivo, assim como descrito na Seção 3.1.4.10.

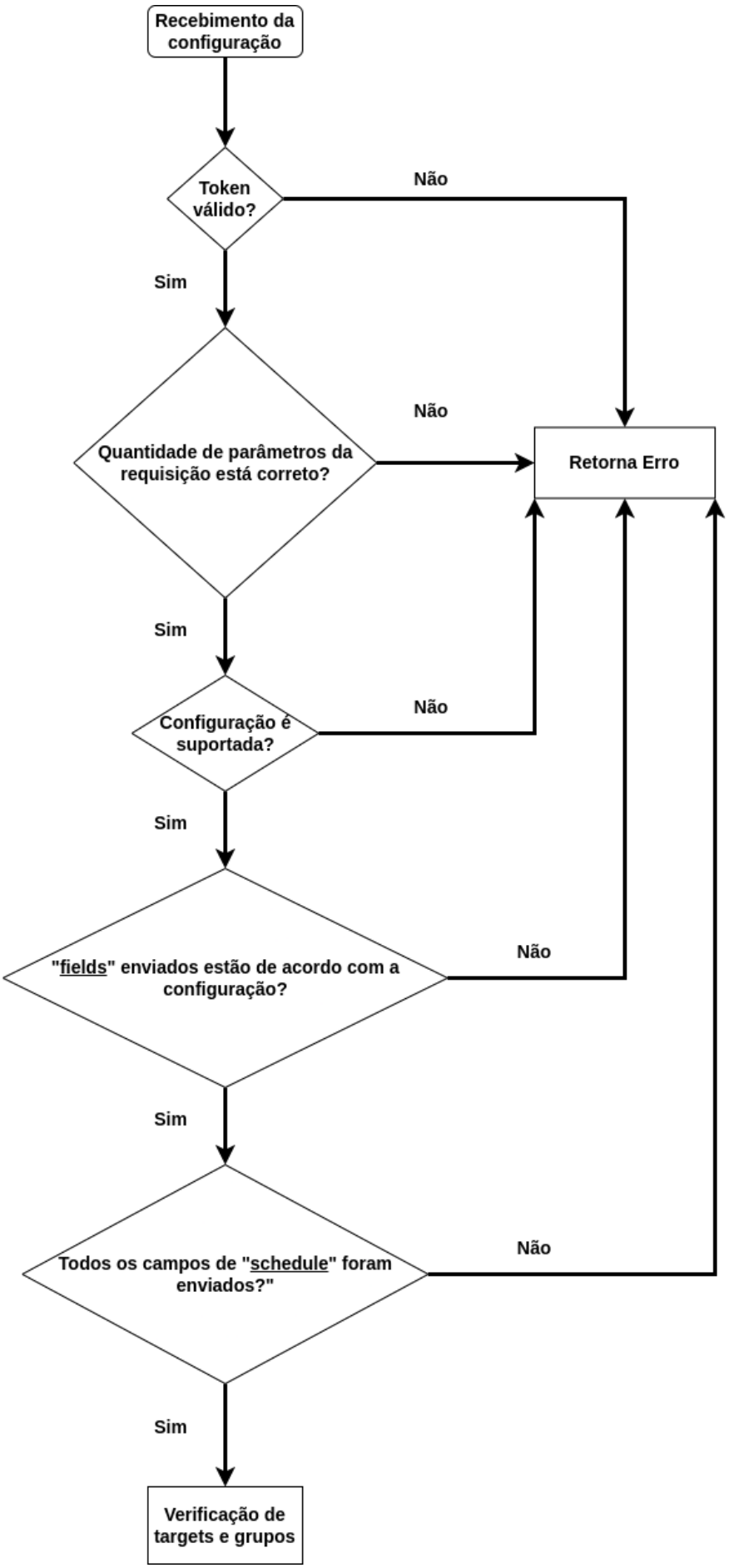

Figura 3.8: Fluxograma da validação de parâmetros de uma requisição no /admin/config.

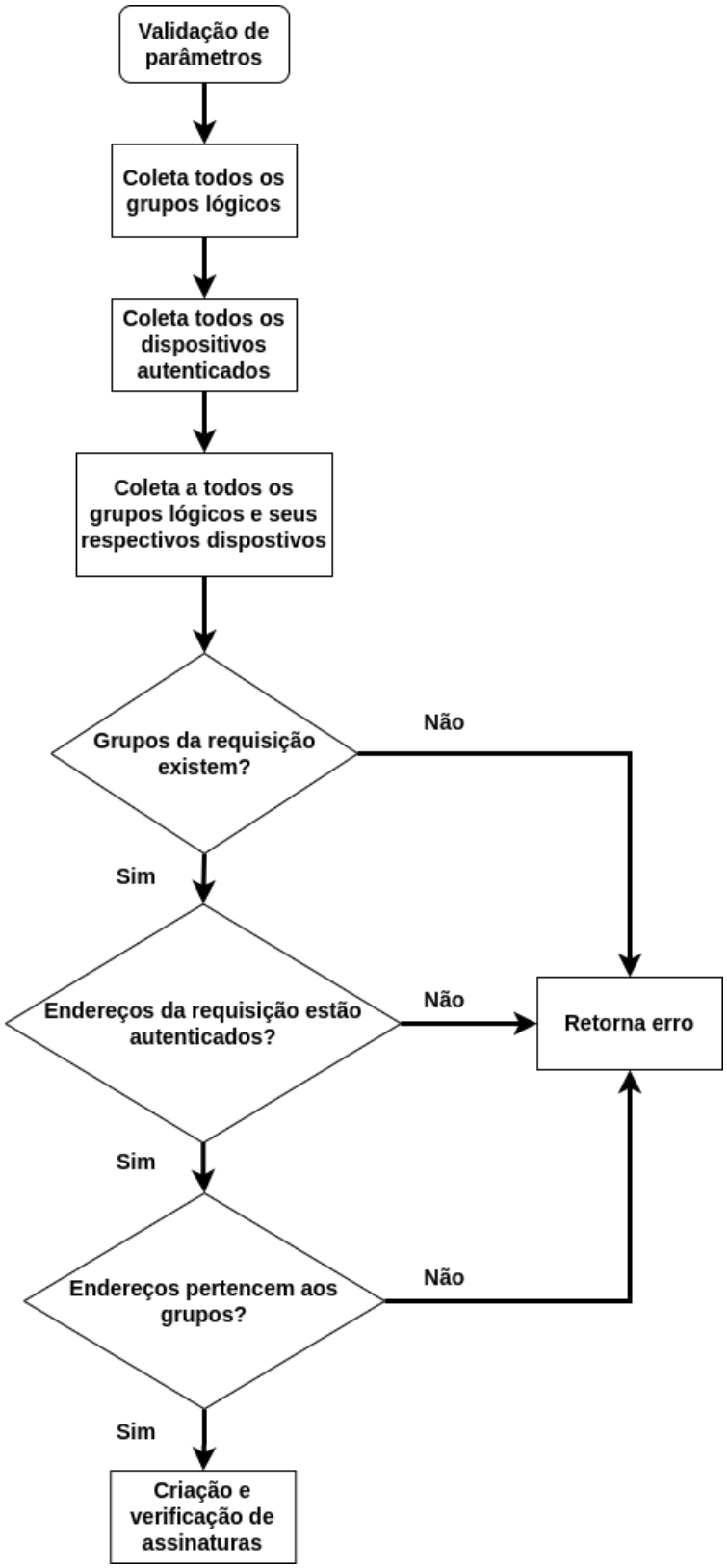

Figura 3.9: Fluxograma da validação targets e grupos lógicos de uma requisição no /admin/-config.

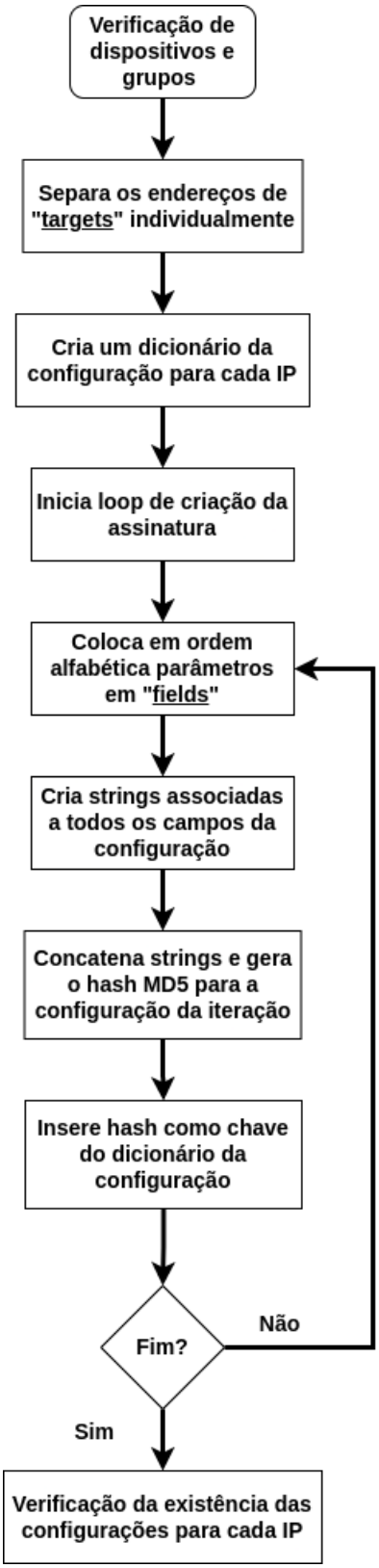

Figura 3.10: Fluxograma da criação de assinaturas para cada IP de uma requisição para /admin/config.

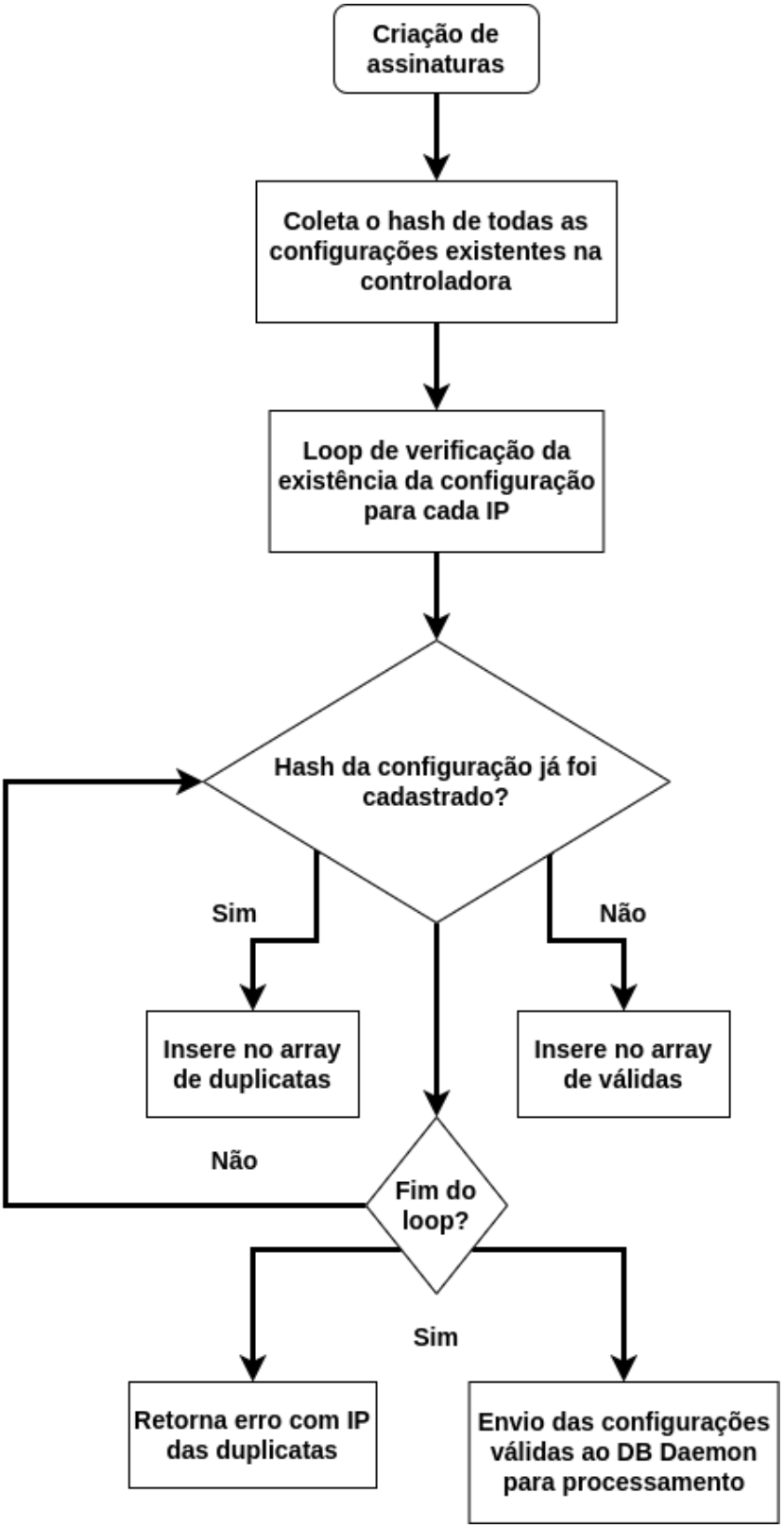

Figura 3.11: Fluxograma da verificação de existência de configuração por assinatura no método /admin/config.

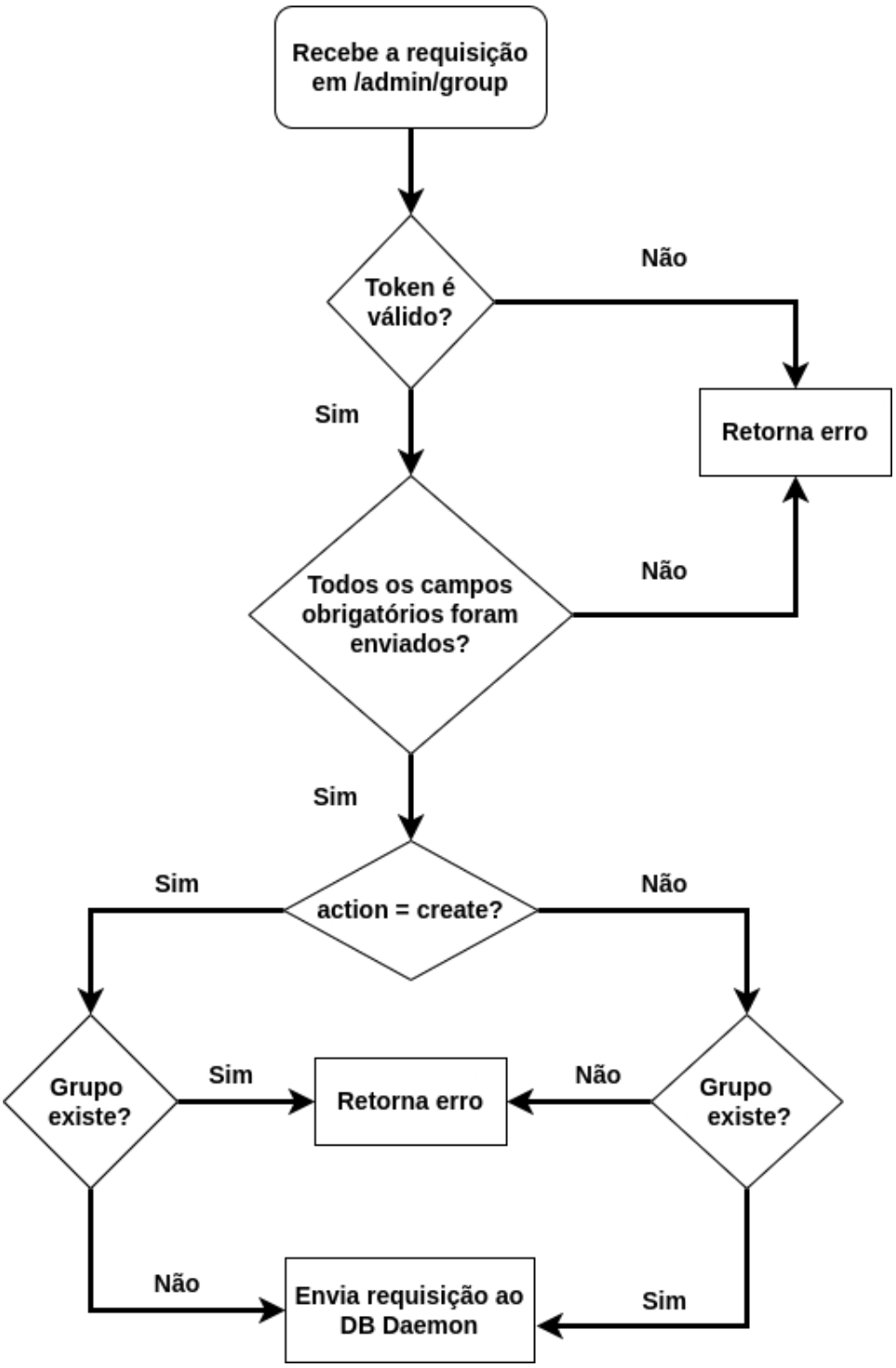

Figura 3.12: Fluxograma de verificação de requisição para o método /admin/group.

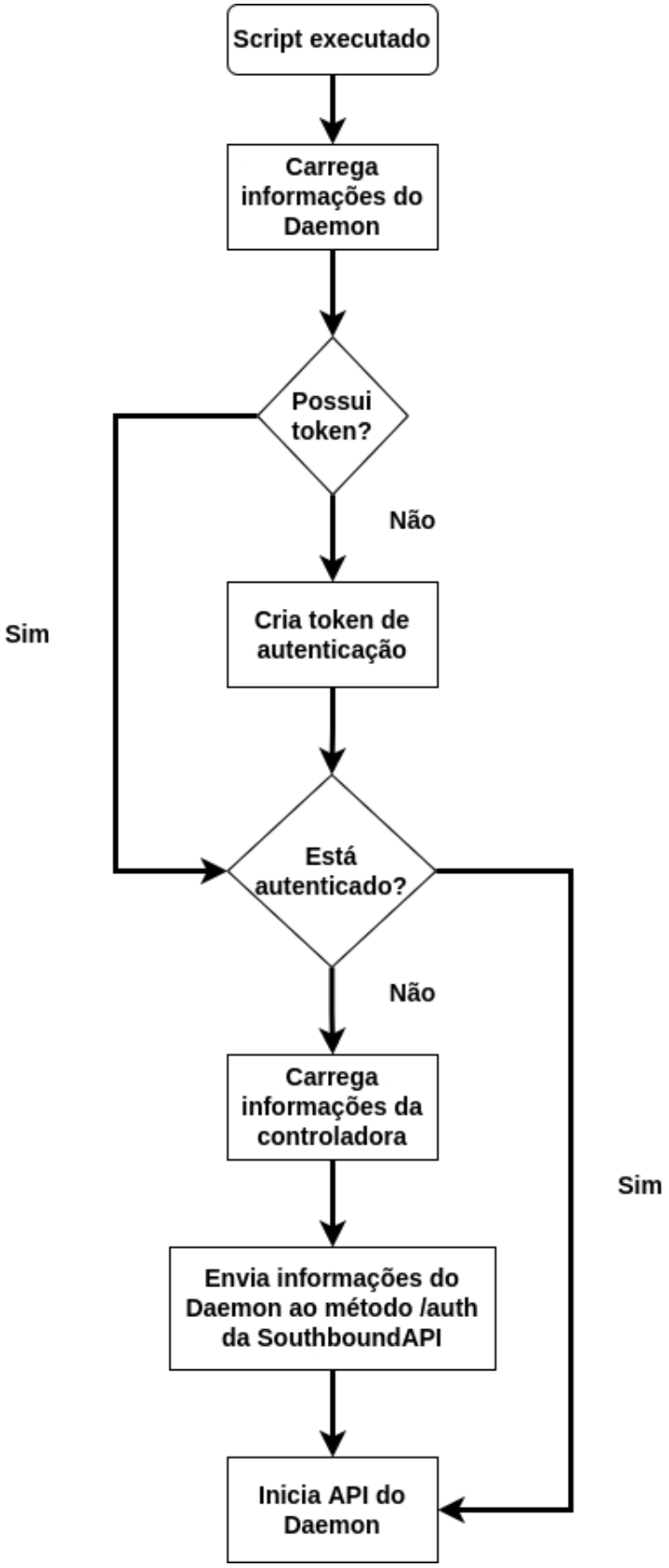

Figura 3.13: Fluxograma de inicialização do OpenWRT-Daemon.

Capítulo 4

Resultados e Análise

Após a implementação da arquitetura, se torna necessário a sua execução e validação em um ambiente similar ao qual ela seria empregada. Para testar a arquitetura implementada nesse trabalho, foi criada uma rede simulada no software GNS3 contendo as entidades da arquitetura, que por sua vez foram usadas para demonstrar algumas das funcionalidades descritas nos capítulos anteriores.

4.1 Ambiente simulado

4.1.1 Visão geral

A arquitetura implementada para os testes pode ser observada na Figura 4.1. Na imagem, pode-se observar uma rede com um ponto central denominado **Router**. Essa entidade é um roteador VyOS que se conecta com quatro redes, sendo elas:

- **NAT (Norte)** - Faz uma NAT com o computador para que a rede simulada consiga acessar a rede externa;
- **192.168.1.0/24 (Oeste)** - Subrede de gerência, contém a máquina do administrador e o OpenWRT-SDN-Controller;
- **192.168.2.0/24 (Leste)** - Subrede da intranet do ambiente, contém um ponto de acesso OpenWRT (**Intranet-AP3**) e os hosts da intranet na subrede *192.168.20.0/24*;
- **192.168.3.0/24 (Sul)** - Subrede de visitantes, contém os pontos de acesso **Guest-AP-Recepcao** e **Guest-AP-PrimeiroAndar**, e os hosts conectados nos APs nas subredes *192.168.10.0/24* e *192.168.30.0/24*, respectivamente.

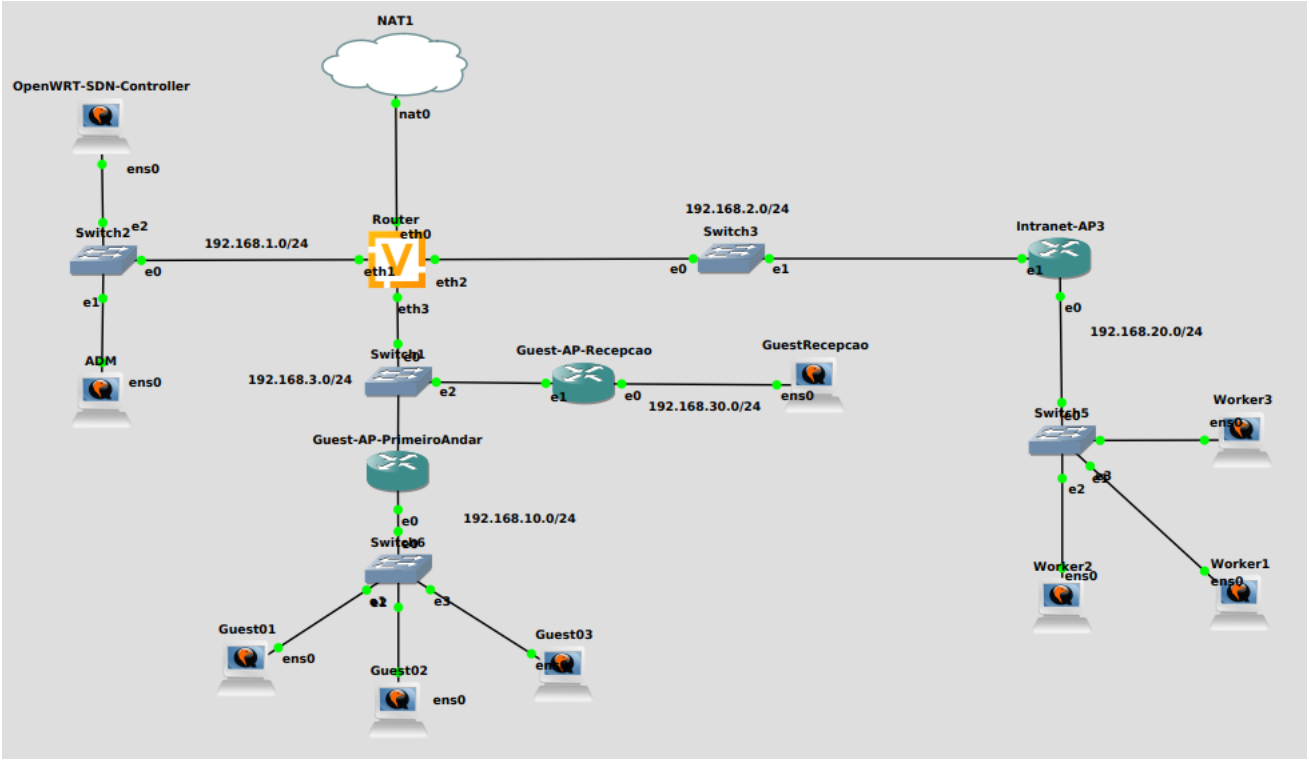

Figura 4.1: Arquitetura implementada para os testes da solução.

4.1.2 Configurações

Algumas configurações foram feitas para que a arquitetura descrita funcione corretamente e todas as entidades estejam conectadas e preparadas para a realização dos testes. As entidades configuradas foram:

- **VyOS Router** - Configurações de IPv4 em interfaces e roteamento entre subredes. Mais detalhes na Seção 4.1.2.1;
- **ADM** - Configurações de IPv4 em interface. Mais detalhes na Seção 4.1.2.2;
- **OpenWRT-SDN-Controller** - Configurações de IPv4 em interface e instalação do controladora no ambiente. Mais detalhes na Seção 4.1.2.3;
- **Dispositivos OpenWRT (GuestAP-PrimeiroAndar, GuestAP-Recepcao e Intranet-AP3)** - Configurações de Ipv4, rotas estáticas, servidor DHCP para os dispositivos conectados e instalação do OpenWRT-Daemon. Mais detalhes na Seção 4.1.2.4;

4.1.2.1 VyOS

O roteador usado para conectar todas a redes presentes na arquitetura está executando o sistema operacional VyOs 1.4. Para que a comunicação entre as subredes funcione corretamente, as configurações presentes na Listagem 4.1 foram executadas no sistema.

Listagem 4.1: Configurações aplicadas no VyOS do ambiente

```
configure
set interfaces ethernet eth0 address dhcp
set interfaces ethernet eth1 address 192.168.1.1/24
set interfaces ethernet eth2 address 192.168.2.1/24
set interfaces ethernet eth3 address 192.168.3.1/24

set nat source rule 100 outbound-interface 'eth0'
set nat source rule 100 source address '192.168.0.0/16'
set nat source rule 100 translation address 'masquerade'

set protocols ospf area 0 network 192.168.1.0/24
set protocols ospf area 0 network 192.168.2.0/24
set protocols ospf area 0 network 192.168.3.0/24
set protocols ospf default-information originate always
set protocols ospf default-information originate metric 10

set protocols ospf default-information originate metric-type
    2
set protocols ospf log-adjacency-changes
set protocols ospf parameters router-id 10.1.1.1
set protocols ospf redistribute connected metric-type 2
set protocols ospf redistribute connected route-map CONNECT

set policy route-map CONNECT rule 10 action permit
set policy route-map CONNECT rule 10 match interface lo

commit
save
exit
```

4.1.2.2 ADM

Para que a máquina do administrador da rede, executando o sistema Ubuntu 21.10, possa se comunicar com as entidades da arquitetura, foi configurado um endereço estático para sua interface. A configuração realizada pode ser observada na Listagem 4.2.

Listagem 4.2: Configuração de IPv4 estático para interface do host ADM.

```

    address - 192.168.1.3
    netmask - 255.255.255.0
    gateway - 192.168.1.1
    dns     - 8.8.8.8
```

4.1.2.3 OpenWRT-SDN-Controller

Para configurar o endereço da interface IPv4 do sistema no qual o OpenWRT-SDN-Controller será executado, foi criado um bash script que é executado durante a inicialização da máquina. O conteúdo do script pode ser observado na Listagem 4.3.

Listagem 4.3: Bash script para a máquina do OpenWRT-SDN-Controller

```
1 #!/bin/bash
2
3 sudo ip link set dev ens3 down
4 sudo ip addr add 192.168.1.2/24 dev ens3
5 sudo ip link set dev ens3 up
6 sudo ip route add default via 192.168.1.1
7 echo 'nameserver 8.8.8.8' > /etc/resolv.conf
```

Além dessas configurações de rede, foi instalado o repositório que contém o código fonte do OpenWRT-SDN-Controller. Após a instalação de bibliotecas Python3 necessárias (requests e flask), foi executado o script *config_cli.py* para configuração dos endereços e portas das entidades que compõe a controladora. As configurações finais da controladora podem ser observadas na Figura 4.2, na qual a porta 8080 representa a *NorthboundAPI*, 8081 a *SouthboundAPI*, 65001 o *DB Daemon* e a 65002 o *Southbound Socket*.

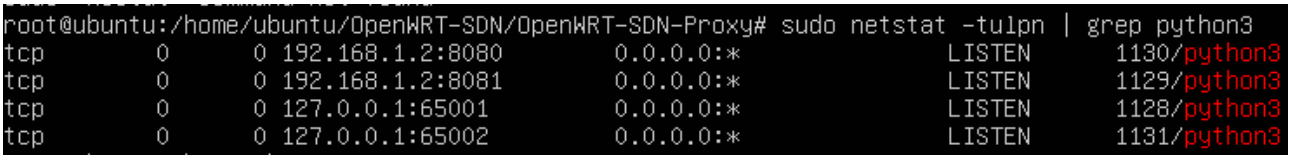

Figura 4.2: Configurações das entidades do OpenWRT-SDN-Controller.

4.1.2.4 Dispositivos OpenWRT

Para cada um dos pontos de acesso no ambiente de testes, foram feitas mudanças em arquivos do sistema para configurar rotas estáticas (Listagem 4.7 no arquivo /etc/config/network), endereço IPv4 da interface WAN (Listagem 4.4 no arquivo /etc/config/network), endereço IPv4 da interface LAN (Listagem 4.5 no arquivo /etc/config/network), DNS oferecido pelo servidor DHCP da interface LAN (Listagem 4.9 no arquivo /etc/config/dhcp), modificação em regra de firewall para permitir o processamento de requisições chegando na interface WAN (Listagem 4.6 no arquivo /etc/config/firewall), instalação do git, do Python3 e suas depedências (Listagem 4.8), e por fim o download e configuração do OpenWRT-Daemon.

Listagem 4.4: Configuração IPv4 interface WAN

```
1
2 config interface 'wan'
3         option device 'eth1'
```

```
4         option proto 'static'
5         option netmask '255.255.255.0'
6         option ipaddr 'address' (
              PrimeiroAndar = 192.168.3.2,
              Recepcao = 192.168.3.3 e
              Intranet = 192.168.2.2)
```

Listagem 4.5: Configuração IPv4 interface LAN

```
1
2 config interface 'lan'
3         option device 'br-lan'
4         option proto 'static'
5         option ipaddr 'address' (
              PrimeiroAndar = 192.168.10.1,
              Recepcao = 192.168.30.1 e
              Intranet = 192.168.20.1)
6         option netmask '255.255.255.0'
7         option ip6assign '60'
```

Listagem 4.6: Configuração de acesso WAN

```
1
2 config zone
3         option name             wan
4         list   network          'wan'
5         list   network          'wan6'
6         option input            ACCEPT
7         option output           ACCEPT
8         option forward          REJECT
9         option masq             1
10        option mtu_fix          1
```

Listagem 4.7: Configuração de rotas estáticas

```
1
2 config route 'route_static'
3         option interface 'wan'
4         option target '0.0.0.0/0'
5         option gateway 'address' (
              PrimeiroAndar e Recepcao =
              192.168.3.1, Intranet =
              192.168.2.1)
```

Listagem 4.8: Instalação de dependências no OpenWRT

```
1
2 opkg update
3 opkg install python3 && opkg install python3
    -pip && pip3 install flask && pip3
    install requests
```

```
opkg remove git
opkg install git-http
opkg install ca-bundle
opkg install qos-scripts
```

Listagem 4.9: Configuração servidor DHCP na LAN

```

config dhcp 'lan'
        option interface 'lan'
        option start '100'
        option limit '150'
        option leasetime '12h'
        option dhcpv4 'server'
        option dhcpv6 'server'
        option ra 'server'
        option ra_slaac '1'
        list ra_flags 'managed-config'
        list ra_flags 'other-config'
        list 'dhcp_option' '6,8.8.8.8'
```

Após a inserção dessas configurações nos arquivos correspondentes, foi necessário executar o script *config_cli.py* para definir qual o endereço e porta que seriam usados durante a execução do OpenWRT-Daemon. Em todos os pontos de acesso foram definidos os endereços da API a partir das configurações IPv4 das interfaces WAN (Listagem 4.4). Além disso, foi configurado o endereço e porta da *SouthboundAPI* como *192.168.1.2* na porta 8081.

Os sistemas diretamente conectados aos pontos de acesso, que representam usuários físicos da rede, estão executando o sistema Ubuntu 21.10 em sua configuração padrão de endereçamento IPv4 por DHCP.

4.2 Demonstração de funcionamento

Afim de demonstrar algumas das funcionalidades descritas no Capítulo 3, serão mostrados o processo de autenticação dos dispositivos, a criação de grupos lógicos, inserção de hosts em grupos lógicos e criação de configurações.

4.2.1 Inicialização e autenticação de OpenWRT-Daemons

O processo de inicialização, descrito na Seção 3.2, realiza a criação dos tokens e faz autenticação dos pontos de acesso executando o OpenWRT-Daemon. A partir da captura dos pacotes que chegaram no host do OpenWRT-SDN-Controller (Figura 4.3), pode-se observar as requisições advindas dos três pontos de acesso executando o OpenWRT-Daemon (Figura

4.4, Figura 4.5 e Figura 4.6). Nos campos das requisições, pode-se observar o envio dos parâmetros necessários para a autenticação no método */auth*, descritos na Seção 3.1.3.2.

8	10.796004	192.168.3.3	192.168.1.2	HTTP	180 POST /auth HTTP/1.1 (application/json)
12	10.802473	192.168.1.2	192.168.3.3	HTTP	92 HTTP/1.1 200 OK (application/json)
22	18.397306	192.168.3.2	192.168.1.2	HTTP	180 POST /auth HTTP/1.1 (application/json)
26	18.404133	192.168.1.2	192.168.3.2	HTTP	92 HTTP/1.1 200 OK (application/json)
35	24.841992	192.168.2.2	192.168.1.2	HTTP	180 POST /auth HTTP/1.1 (application/json)
39	24.846384	192.168.1.2	192.168.2.2	HTTP	92 HTTP/1.1 200 OK (application/json)
48	25.896254	192.168.1.3	35.232.111.17	HTTP	153 GET / HTTP/1.1
50	26.100097	35.232.111.17	192.168.1.3	HTTP	214 HTTP/1.1 204 No Content

Figura 4.3: Requisições enviadas para autenticação dos pontos de acesso.

```
▸ Internet Protocol Version 4, Src: 192.168.3.3, Dst: 192.168.1.2
▸ Transmission Control Protocol, Src Port: 49310, Dst Port: 8081, Seq: 206, Ack: 1, Len: 114
▸ [2 Reassembled TCP Segments (319 bytes): #7(205), #8(114)]
▸ Hypertext Transfer Protocol
▾ JavaScript Object Notation: application/json
  ▾ Object
    ▾ Member Key: address
        String value: 192.168.3.3
        Key: address
    ▾ Member Key: port
        Number value: 50000
        Key: port
    ▾ Member Key: netmask
        String value: 255.255.255.0
        Key: netmask
    ▾ Member Key: token
        String value: 9a89364e271b2406f1f25e182c3ade2b
        Key: token
```

Figura 4.4: Requisição de autenticação do Guest-AP-Recepcao.

```
▸ Internet Protocol Version 4, Src: 192.168.3.2, Dst: 192.168.1.2
▸ Transmission Control Protocol, Src Port: 36020, Dst Port: 8081, Seq: 206, Ack: 1, Len: 114
▸ [2 Reassembled TCP Segments (319 bytes): #21(205), #22(114)]
▸ Hypertext Transfer Protocol
▾ JavaScript Object Notation: application/json
  ▾ Object
    ▾ Member Key: address
        String value: 192.168.3.2
        Key: address
    ▾ Member Key: port
        Number value: 50000
        Key: port
    ▾ Member Key: netmask
        String value: 255.255.255.0
        Key: netmask
    ▾ Member Key: token
        String value: 54cd26a55e72620710c87eaddd319344
        Key: token
```

Figura 4.5: Requisição de autenticação do Guest-AP-PrimeiroAndar.

A partir da máquina ADM, foi enviada uma requisição para o método */admin/list/host* da *NorthboundAPI* no intuito de validar a autenticação de dispositivos gerenciados da perspectiva de um administrador. Na Figura 4.7 pode-se observar o resultado da requisição HTTP GET no método.

4.2.2 Criação de grupos lógicos

Diante da própria topologia da arquitetura e da descrição das subredes, serão criados dois grupos lógicos para segmentar os dispositivos gerenciados. Os grupos serão nomeados de

```
▸ Internet Protocol Version 4, Src: 192.168.2.2, Dst: 192.168.1.2
▸ Transmission Control Protocol, Src Port: 49456, Dst Port: 8081, Seq: 206, Ack: 1, Len: 114
▸ [2 Reassembled TCP Segments (319 bytes): #34(205), #35(114)]
▸ Hypertext Transfer Protocol
▾ JavaScript Object Notation: application/json
  ▾ Object
    ▾ Member Key: address
        String value: 192.168.2.2
        Key: address
    ▾ Member Key: port
        Number value: 50000
        Key: port
    ▾ Member Key: netmask
        String value: 255.255.255.0
        Key: netmask
    ▾ Member Key: token
        String value: b74f7e814cce4dcf4e189d5e65e5dfe0
        Key: token
```

Figura 4.6: Requisição de autenticação do Intranet-AP3.

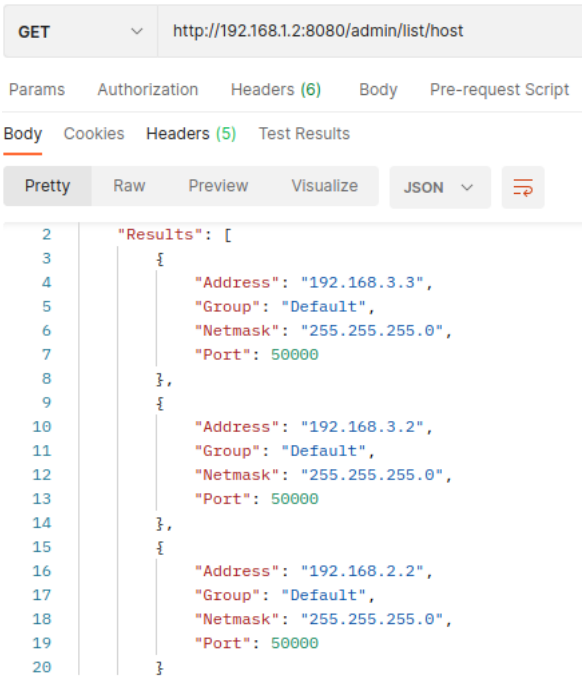

Figura 4.7: Requisição de requisição ao método /admin/list/host.

Guests e *Intranet*. As requisições, enviadas pela máquina ADM, serão baseadas no formato descrito na Seção 3.1.2.3.

Pode-se observar na Figura 4.8 os pacotes HTTP capturados durante o envio das requisições de criação dos grupos. As Figuras 4.9 e 4.10 mostram em mais detalhes o corpo das requisições enviadas. Para validar a criação dos grupos lógicos, a Figura 4.11 demonstra o envio de uma requisição HTTP GET ao método */admin/list/group* da *NorthboundAPI*.

```
 5 7.379182   192.168.1.3   192.168.1.2   HTTP   448 POST /admin/group HTTP/1.1  (text/plain)
 8 7.393593   192.168.1.2   192.168.1.3   HTTP   128 HTTP/1.1 200 OK  (application/json)
18 16.756738  192.168.1.3   192.168.1.2   HTTP   450 POST /admin/group HTTP/1.1  (text/plain)
21 16.771303  192.168.1.2   192.168.1.3   HTTP   128 HTTP/1.1 200 OK  (application/json)
```

Figura 4.8: Requisições enviadas ao método /admin/group.

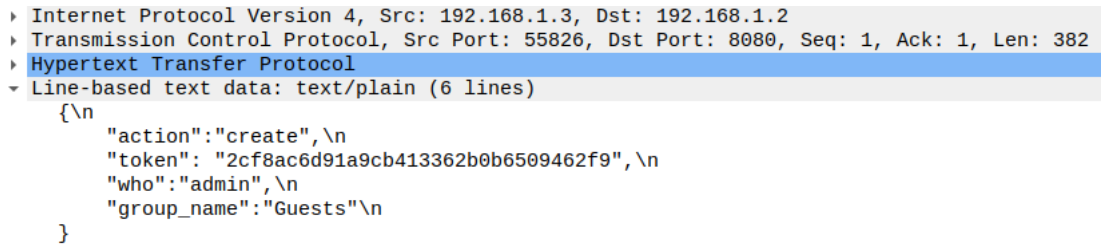

Figura 4.9: Requisição de criação do grupo Guests.

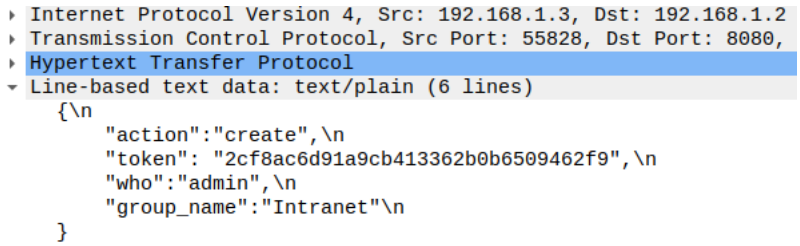

Figura 4.10: Requisição de criação do grupo Intranet.

4.2.3 Inserção de OpenWRTs em grupos lógicos

Após a criação dos grupos lógicos, basta inserir os dispositivos gerenciados autenticados na controladora nas segmentações que fazem mais sentido. No caso da topologia usada para a demonstração desses resultados, os dispositivos Guest-AP-Recepcao e Guest-AP-PrimeiroAndar serão inseridos no grupo Guests, enquanto o ponto de acesso Intranet-AP3 será inserido no grupo Intranet. As requisições enviadas ao método */admin/host* da *NorthboundAPI* respeitam o formato descrito na Seção 3.1.2.4.

Pode-se observar na Figura 4.12 os pacotes HTTP capturados durante o envio das requisições para inserção de dispositivos autenticados em grupos lógicos existentes na controladora. As Figuras 4.13 e 4.14 mostram em mais detalhes as requisições enviadas. Para validar a inserção dos dispositivos autenticados nos grupos lógicos, a Figura 4.15 demonstra o envio de uma requisição HTTP GET ao método */admin/list/host* da *NorthboundAPI* no intuito de mostrar a atualização dos grupos observados na Figura 4.7.

4.2.4 Criação de configurações

Para demonstrar as funcionalidades da solução, serão criadas configurações que afetam diretamente o processamento de pacotes pelo OpenWRT no seu Firewall e no seu serviço de QoS.

4.2.4.1 Regras de firewall

Diante da segmentação entre as redes do ambiente, conceitualmente não faz sentido uma sub-rede associada a um ponto de acesso do grupo *Guests* se comunicar com um dispositivo

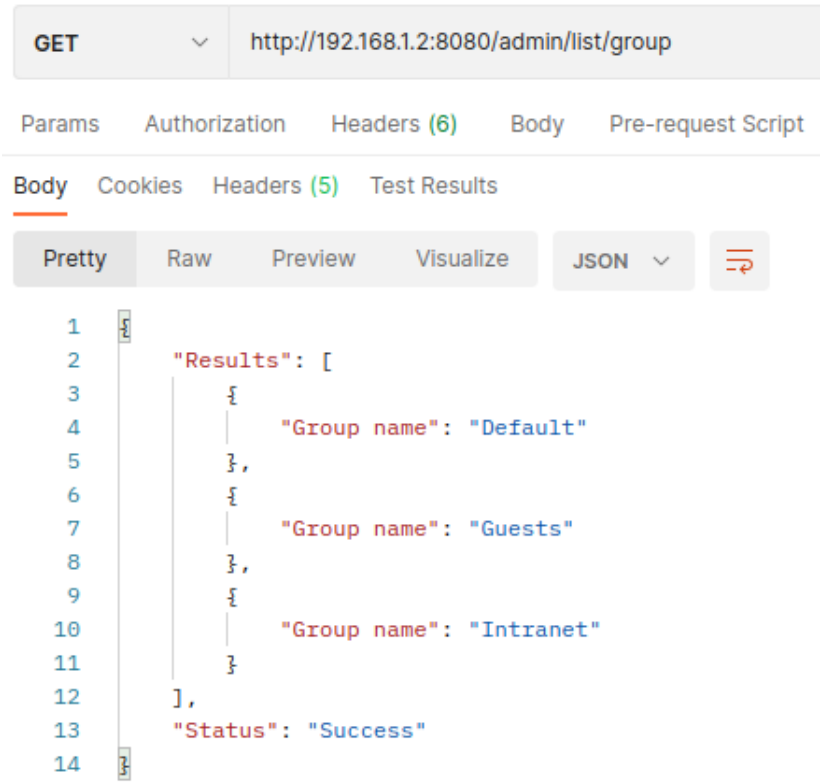

Figura 4.11: Resultado de requisição ao método /admin/list/group.

```
36 62.037713    192.168.1.3    192.168.1.2    HTTP    479 POST /admin/host HTTP/1.1  (text/plain)
39 62.041794    192.168.1.2    192.168.1.3    HTTP    240 HTTP/1.1 200 OK  (application/json)
49 73.690071    192.168.1.3    192.168.1.2    HTTP    480 POST /admin/host HTTP/1.1  (text/plain)
52 73.705110    192.168.1.2    192.168.1.3    HTTP    140 HTTP/1.1 200 OK  (application/json)
```

Figura 4.12: Requisições enviadas ao método /admin/host.

da *Intranet*.

Para bloquear qualquer tipo de comunicação entre a subrede 192.168.10.0/24 (usuários do ponto de acesso Guest-AP-PrimeiroAndar) e endereços da *Inranet 192.168.2.0/24*, foi enviada uma configuração para o método */admin/config* da controladora criando uma configuração de firewall para bloqueio desse tipo de tráfego no Guest-AP-PrimeiroAndar. A captura da requisição com os parâmetros enviados pode ser observada na Figura 4.16, enquanto o resultado em um host pertencente a sub-rede de visitantes bloqueada pode ser observado na Figura 4.17.

4.2.4.2 Regras de QoS

Para demonstrar a funcionalidades de QoS, foi enviada uma requisição para o método */admin/config* da controladora no intuito de criar uma definição de QoS para a interface WAN do Guest-AP-PrimeiroAndar. A configuração, que pode ser observada na Figura 4.18, teve como objetivo colocar a taxa de transmissão máxima da interface WAN para **Download** = 10 kbps e **Upload** = 5 kbps. Na Figura 4.19 pode-se observar um gráfico de I/O da interface WAN após a aplicação da configuração, em que houve uma queda muito brusca do envio de

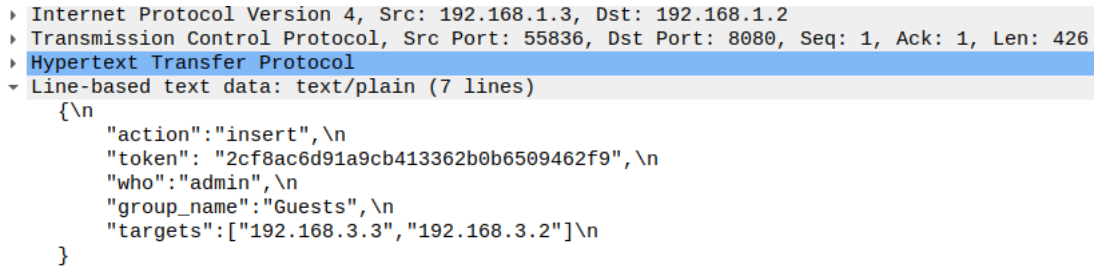

Figura 4.13: Requisição para inserção dos pontos de acesso no grupo Guests.

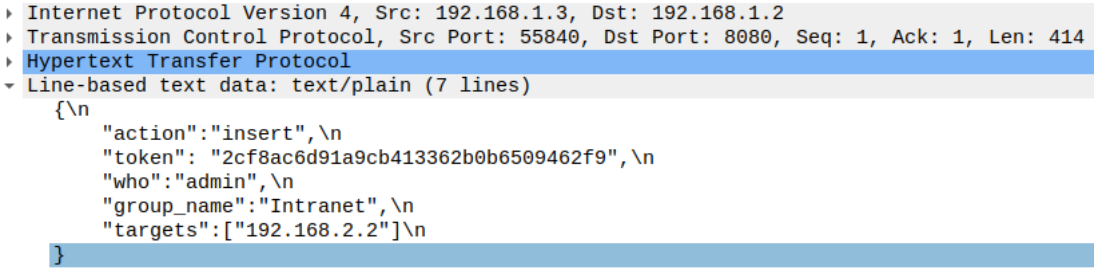

Figura 4.14: Requisição para inserção do ponto de acesso no grupo Intranet.

pacotes após a aplicação da configuração de um limite de taxa de transmissão muito baixo. O tráfego anterior a queda foi gerado por um host conectado a ele pela rede *192.168.10.0/24* ao asstir uma livestream no Youtube.

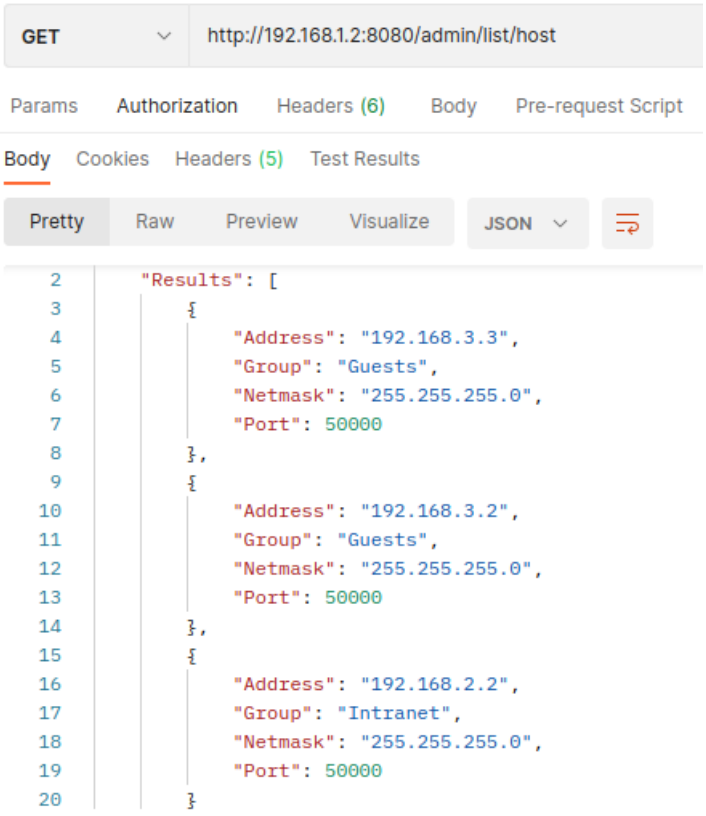

Figura 4.15: Nova requisição para método /admin/list/host.

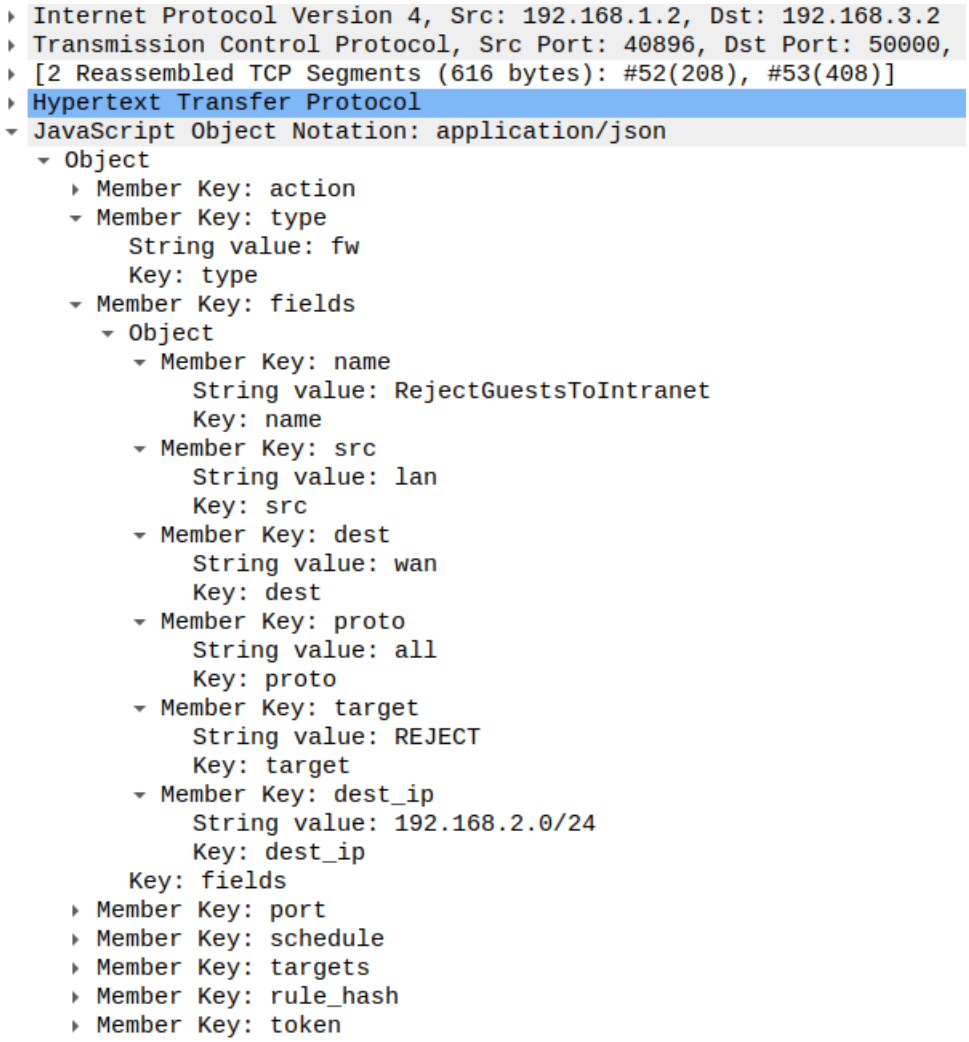

Figura 4.16: Requisição para criação para criação de regra para bloqueio de tráfego no Guest-AP-Primeiro-Andar.

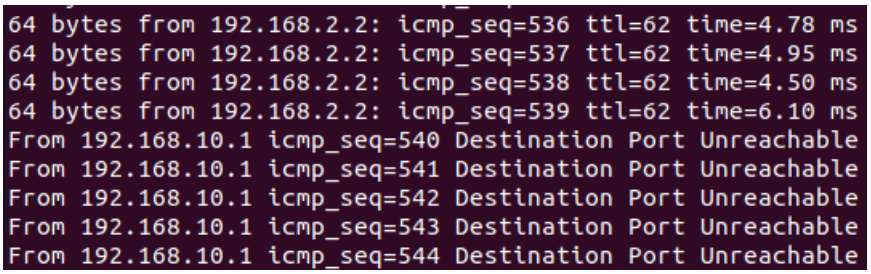

Figura 4.17: Bloqueio de comunicação entre hosts conectados ao Guest-AP-PrimeiroAndar e a Intranet.

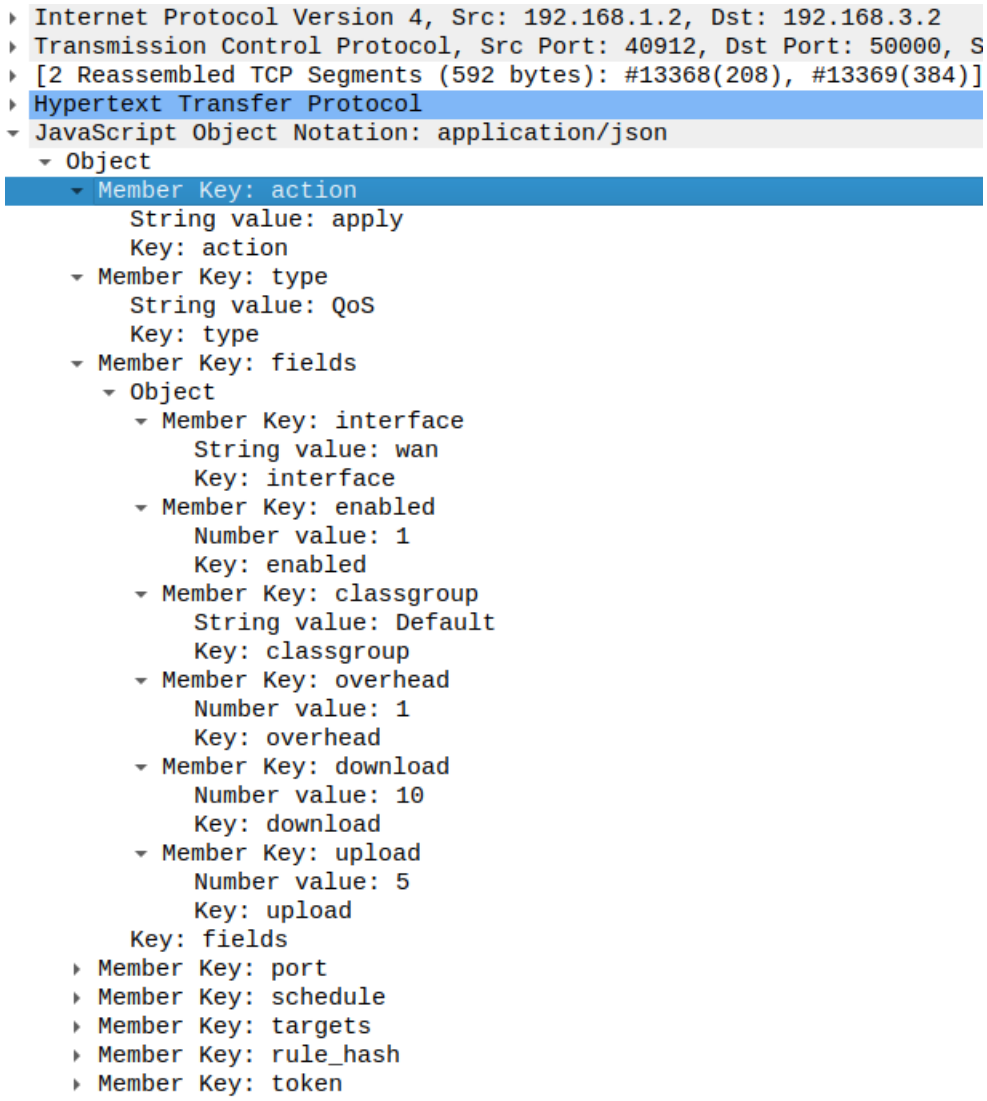

Figura 4.18: Requisição para criação de configuração de QoS no Guest-AP-PrimeiroAndar.

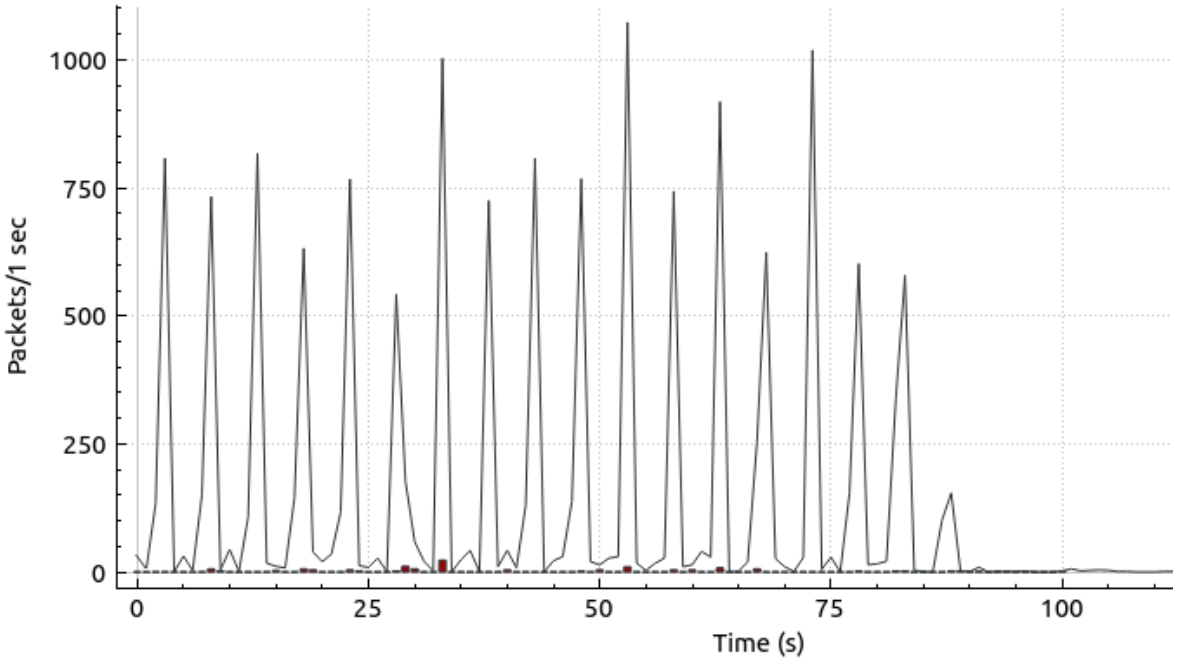

Figura 4.19: Gráfico de I/O da interface WAN do Guest-AP-PrimeiroAndar.

Conclusão

A arquitetura implementada neste trabalho dispõe das funcionalidades necessárias para proporcionar uma experiência básica de gerenciamento a administradores que a utilizem em seu ambiente. A maneira pela qual as entidades foram feitas tornou possível o recebimento de requisições para listagem de grupos, de dispositivos autenticados e de configurações existentes. Além disso, permite a criação de grupos lógicos, a inserção e remoção de dispositivos em grupos lógicos e a criação ou remoção de configurações para os tipos suportados pela controladora.

Desde a definição de seus objetivos, o trabalho teve como propósito a criação de uma base para uma solução que possa ser ampliada. As entidades que compõem essa implementação podem ser facilmente modificadas para ampliação de suas funcionalidades no intuito de tornar a arquitetura ainda mais robusta e completa para todas as necessidades que um administrador de uma rede empresarial possua.

Por fim, o capítulo de resultados demonstra que as entidades criadas para o funcionamento da arquitetura (OpenWRT-Controller e OpenWRT-Daemon) foram capazes de entregar as capacidades que se propuseram a oferecer.

Trabalhos futuros: Assim como dito anteriormente, esse trabalho tem como objetivo servir como base para uma solução mais abrangente para gerenciamento de dispositivos OpenWRT por uma controladora centralizada. Para isso, são planejadas funcionalidades como uma interface de gerenciamento web, coleta de métricas de recursos dos dispositivos e ampliação das configurações suportadas para abordar todas as que são possíveis no sistema.

Referências Bibliográficas

[Fainelli 2008] Fainelli, F. (2008). The openwrt embedded development framework. In *Proceedings of the Free and Open Source Software Developers European Meeting*, page 106. sn.

[Hi Châu Lê 2020] Hi Châu Lê, K.-T. N. (2020). Development Of SDN-Based Wi-Fi Ap Using Openwrt And Raspberry Pi 3. *Journal of Science and Technology on Information and Communications*.

[Manish Paliwal 2018] Manish Paliwal, Deepti Shrimankar, O. T. (2018). Controllers in SDN: A Review Report. *IEEE Access*.

[Maurizio Goretti 2012] Maurizio Goretti, Davide Guerri, F. L. (2012). ProvinciaWiFi: A 1000 hotspot free, public, open source WiFi network. *International Telecommunications Network Strategy and Planning Symposium (NETWORKS)*.

[Muxi Yan 2016] Muxi Yan, Jasson Casey, P. S. A. S. (2016). ÆtherFlow: Principled Wireless Support in SDN. *Internation Conference on Network Protocols (ICNP)*.

[Nick Feamster 2013] Nick Feamster, Jennifer Rexford, E. Z. (2013). The Road to SDN: An Intellectual History of Programmable Networks. *Princeton University*.

[OpenNetworkFoundation 2012] OpenNetworkFoundation (2012). Openflow switch specification. `https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf`. (Accessed on 10/09/2022).

[OpenNetworkFoundation 2013] OpenNetworkFoundation (2013). Software-defined networking: The new norm for networks. `https://opennetworking.org/sdn-resources/whitepapers/software-defined-networking-the-new-norm-for-networks/`. (Accessed on 10/09/2022).

[OpenWISP-Architecture] OpenWISP-Architecture. Well-known users of sqlite. `https://openwisp.io/docs/general/architecture.html`. (Accessed on 14/09/2022).

[OpenWRT-About 2016] OpenWRT-About (2016). About openwrt. `https://openwrt.org/about`. (Accessed on 13/09/2022).

[OpenWRT-Config 2016] OpenWRT-Config (2016). Openwrt configurations. https://openwrt.org/docs/guide-user/start. (Accessed on 13/09/2022).

[POX-Controller] POX-Controller. Pox. https://github.com/noxrepo/pox. (Accessed on 15/09/2022).

[Raphael Horvath 2015] Raphael Horvath, Dietmar Nedbal, M. S. (2015). A Literature Review on Challenges and Effects of Software Defined Networking. *Conference on ENTERprise Information Sysmtems (CENTERIS)*.

[SQLite-About] SQLite-About. About sqlite. https://www.sqlite.org/about.html. (Accessed on 14/09/2022).

[SQLite-High-Profile] SQLite-High-Profile. Well-known users of sqlite. https://www.sqlite.org/famous.html. (Accessed on 14/09/2022).

[SQLite-Most-Used] SQLite-Most-Used. Most widely deployed and used database engine. https://www.sqlite.org/mostdeployed.html. (Accessed on 14/09/2022).

[Wenfeng Xia 2015] Wenfeng Xia, Yonggang We, C. H. F. D. N. H. X. (2015). A Survey on Software-Defined Networking. *IEEE Communications Surveys Tutorials*.

Apêndice

Código fonte: O código desenvolvido para esse projeto, juntamente com todos seus scripts e documentos descrevendo os formatos de requisição, se encontram no repositório público de códigos Github acessível pelo endereço: *https://github.com/Danielvenzi/OpenWRT-SDN*. O uso da solução deve ser feito a partir das descrições passadas neste trabalho.