UNIVERSIDADE FEDERAL DO PAMPA

MAURÍCIO MARTINUZZI FIORENZA

GERENCIAMENTO DE FIREWALLS EM REDES HÍBRIDAS

Alegrete
2021

MAURÍCIO MARTINUZZI FIORENZA

GERENCIAMENTO DE FIREWALLS EM REDES HÍBRIDAS

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Software como requisito parcial para a obtenção do título de Mestre em Engenharia de Software.

Orientador: Diego Kreutz
Coorientador: Rodrigo Brandão Mansilha

Alegrete
2021

Ficha catalográfica elaborada automaticamente com os dados fornecidos pelo(a) autor(a) através do Módulo de Biblioteca do Sistema GURI (Gestão Unificada de Recursos Institucionais).

F518g Fiorenza, Maurício Martinuzzi

Gerenciamento de Firewalls em Redes Híbridas / Maurício Martinuzzi Fiorenza.

80 f.: il.

Orientador: Diego Kreutz
Coorientador: Rodrigo Brandão Mansilha
Dissertação (Mestrado Profissional) - Universidade Federal do Pampa, Programa de Pós-Graduação em Engenharia de Software, 2021.

1. Políticas de segurança. 2. Gerenciamento de firewalls. 3. Redes híbridas. 4. Arquitetura de software. 5. Linguagem. 6. Intenções. I. Título.

SERVIÇO PÚBLICO FEDERAL
MINISTÉRIO DA EDUCAÇÃO
Universidade Federal do Pampa

MAURÍCIO MARTINUZZI FIORENZA

GERENCIAMENTO DE FIREWALLS EM REDES HÍBRIDAS

Dissertação/Tese apresentada ao Programa de Pós-Graduação em Engenharia de Software da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Mestre em Engenharia de Software.

Dissertação defendida e aprovada em: 10 de março de 2021.

Banca examinadora:

_____

Prof. Dr. Diego Kreutz
Orientador
Unipampa

Prof. Dr. Rodrigo Brandão Mansilha (Coorientador)
Unipampa

Prof. Dr. Douglas Dyllon Jeronimo de Macedo
UFSC

Prof. Dr. Eduardo Luzeiro Feitosa
UFAM

Prof. Dr. Roger Kreutz Immich
UFRN

Assinado eletronicamente por **RODRIGO BRANDAO MANSILHA**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 15/04/2021, às 11:58, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **DIEGO LUIS KREUTZ**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 15/04/2021, às 15:59, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **DOUGLAS DYLLON JERONIMO DE MACEDO**, **Usuário Externo**, em 15/04/2021, às 16:02, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **Roger Kreutz Immich**, **Usuário Externo**, em 15/04/2021, às 16:05, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **Eduardo Luzeiro Feitosa**, **Usuário Externo**, em 15/04/2021, às 16:08, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

A autenticidade deste documento pode ser conferida no site

https://sei.unipampa.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **0502889** e o código CRC **B13FD0B4**.

RESUMO

O gerenciamento de firewalls é um processo desafiador. A configuração incorreta desse tipo de equipamento pode gerar desde brechas de segurança que comprometam as informações trafegadas, até a indisponibilidade de serviços críticos. O processo é especialmente desafiador redes híbridas, que combinam tecnologias convencionais e emergentes, como SDN. O administrador de firewalls de uma rede híbrida precisa conhecer uma combinação potencialmente extensa de vocabulário, sintaxes, semânticas e processos para configurar os firewalls conforme sua intenção. Para facilitar o trabalho dos administradores de redes híbridas, propomos uma arquitetura de software, denominada FWunify, e uma linguagem de representação de intenções de segurança, denominada FWlang. A arquitetura de software visa orientar o desenvolvimento de soluções para gerenciamento de firewalls em redes híbridas através de camadas e módulos que facilitam a manutenabilidade e extensibilidade da solução. Por exemplo, um módulo de tradução de comandos para determinado firewall pode ser adicionado (ou removido) da solução impactando minimamente apenas nas camadas adjacentes. A linguagem FWlang visa simplificar e reduzir o vocabulário necessário para gerenciar um parque híbrido de firewalls através de intenções, que permitem representar configurações de segurança em um nível mais alto e amigável de abstração que o nível de uma política convencional. Para verificar a viabilidade técnica da solução, implementamos uma instância da arquitetura de software FWunify, incluindo a FWlang. Uma avaliação funcional abrangente da FWunify demonstra a sua acurácia de tradução (para comandos de baixo nível utilizados em firewalls) e a sua capacidade de gerenciar políticas em firewalls reais. Através de uma avaliação baseada em *datasets* reais, demonstramos que a FWlang pode propiciar uma redução de até 72% no número de termos utilizados para descrever um determinado tipo de políticas. Por fim, avaliamos aspectos subjetivos, como intuitividade e complexidade, da FWlang através de um questionário respondido por administradores de sistemas.

Palavras-chave: Políticas de segurança. gerenciamento de firewalls. redes híbridas. arquitetura de software. linguagem. intenções.

ABSTRACT

Managing firewalls is a challenging process. The incorrect configuration of this type of equipment can generate problems ranging from security breaches that compromise the information transmitted to the unavailability of critical services. The process is especially challenging for hybrid networks, which combine conventional and emerging technologies such as SDN. The firewall administrator of a hybrid network needs to know a potentially extensive combination of vocabulary, syntax, semantics and processes to configure the firewalls according to their intention. To facilitate the work of hybrid network administrators, we propose a software architecture, called FWunify, and a language for representing security intentions, called FWlang. The software architecture aims to guide the development of solutions for managing firewalls in hybrid networks through layers and modules that facilitate the maintainability and extensibility of the solution. For example, a command translation module for a given firewall can be added (or removed) from the solution with minimal impact only on adjacent layers. The FWlang language aims to simplify and reduce the vocabulary needed to manage a hybrid park of firewalls through intentions, which allow representing security configurations at a higher and more friendly level of abstraction than the level of a conventional policy. To verify the technical feasibility of the solution, we implemented an instance of the FWunify software architecture, including FWlang. A comprehensive functional assessment of FWunify demonstrates its accuracy for translating (for low-level commands used in firewalls) and its ability to manage policies across real firewalls. Through an assessment based on real datasets, we demonstrate that FWlang can provide a reduction of up to 72% in the number of terms used to describe a given type of policy. Finally, we evaluate subjective aspects, such as intuitiveness and complexity, of FWlang through a questionnaire answered by system administrators.

Keywords: Security policies. firewall management. hybrid networks. software architecture. language. intents.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACL	Access Control List
API	Application Programming Interface
ASA	Adaptive Security Appliances
ASDM	Adaptive Security Device Manager
CAPEX	Capital Expenditure
CIDR	Classless InterDomain Routing
CLI	Command-line Interface
DMZ	Demilitarized Zone
EBNF	Extended Backus–Naur Form
ForCES	Forwarding and Control Element Separation
FTP	File Transfer Protocol
Gbps	Gigabits por segundo
GNS3	Graphical Network Simulator 3
HTTP	Hypertext Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
IBF	Intent-based Firewalling
IBN	Intent-based Networking
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPS	Intrusion Prevention System
Kbps	Kilobits por segundo
Mbps	Megabits por segundo

NAT	Network Address Translation
NFV	Network Functions Virtualization
NGFW	Next Generation Firewalls
NIDS	Network Intrusion Detection System
OPEX	Operational Expenditure
P4	Programming Protocol-independent Packet Processors
PIX	Private Internet eXchange
POF	Protocol Oblivious Forwarding
RBAC	Role-based Access Control
REST	Representational State Transfer
RPC	Remote Procedure Call
SDN	Software-Defined Networking
SDNs	Software-Defined Networks
SMTP	Simple Mail Transfer Protocol
SSH	Secure Socket Shell
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
URL	Uniform Resource Locator
YAML	Yet Another Markup Language
WAN	Wide Area Network

SUMÁRIO

1 INTRODUÇÃO

As redes híbridas representam o cenário de adoção progressiva de equipamentos baseados nos padrões e protocolos do paradigma SDN (*Software-Defined Networking*) como OpenFlow, P4 (*Programming Protocol-independent Packet Processors*), ForCES (*Forwarding and Control Element Separation*) e POF (*Protocol Oblivious Forwarding*) (MCKEOWN et al., 2008; BOSSHART et al., 2014; YANG et al., 2004; SONG, 2013; KREUTZ et al., 2014). Uma rede híbrida é caracterizada como uma rede que contém equipamentos de rede tradicionais (*i.e.*, comutadores Ethernet com uma arquitetura verticalmente integrada (RAMOS; KREUTZ; VERISSIMO, 2015; KREUTZ et al., 2014)) e equipamentos de rede SDN. As redes híbridas são atrativas devido a fatores como investimento de capital (CAPEX - *Capital Expenditure*) e formação de recursos humanos e demais custos operacionais (OPEX - *Operational Expenditure*) graduais (ABDALLAH et al., 2017). Entretanto, elas suscitam desafios, como o gerenciamento integrado de soluções de redes tradicionais e SDN (VISSICCHIO; VANBEVER; BONAVENTURE, 2014; SANDHYA; SINHA; HARIBABU, 2017).

Os desafios de integração e interoperabilidade de soluções híbridas também ocorrem no âmbito de firewalls. Uma rede híbrida pode incorporar diversas soluções de firewall, tanto tradicionais (*e.g.*, IPTables, pfSense, Cisco, Palo Alto e Check Point) como SDN (*e.g.*, baseadas em OpenFlow e P4). Fabricantes como Cisco, Palo Alto e Check Point oferecem soluções de gerenciamento específicas para suas famílias de equipamentos. Por exemplo, a Cisco fornece o software *Cisco Firepower Management Center*[1], que permite gerenciar as linhas Firepower de NGFWs (*Next Generation Firewalls*), mas não suporta outras linhas de firewalls da própria empresa ou de terceiras. O mesmo problema ocorre com soluções voltadas para SDN que aplicam políticas de segurança através de recursos providos por controladores SDN (*e.g.*, Floodlight e POX) (FIESSLER et al., 2018; OTHMAN et al., 2017; ZKIK; HAJJI; ORHANOU, 2019; MORZHOV; ALEKSEEV; NIKITINSKIY, 2016) ou diretamente nos equipamentos utilizando padrões e protocolos como o OpenFlow e P4 (CAPROLU; RAPONI; PIETRO, 2019; VÖRÖS; KISS, 2016; Vinh Tran; Ahn, 2016). Este é o caso, por exemplo, do P4Guard (DATTA et al., 2018), que implementa e gerencia políticas de segurança exclusivamente em ambientes P4.

[1]<https://www.cisco.com/c/en/us/products/security/firepower-management-center/index.html>

1.1 Problema

Equipamentos de firewalls podem variar em termos de fabricante, linha e modelo, e essas variações tendem a refletir nas linguagens ou versões de linguagens de configuração. Normalmente, as linguagens de configuração variam entre fabricantes. Por exemplo, ao descrevermos uma mesma política para um firewall Cisco ASA (*Adaptive Security Appliance*) e para um firewall IPTables, as semelhanças na terminologia utilizada se resumem aos valores dinâmicos (*i.e*, IP (*Internet Protocol*) de origem, IP de destino e prioridade da regra). Os demais termos, das sintaxes específicas, são diferentes para cada solução de firewall. Essas diferenças aumentam significativamente na medida que outras soluções são agregadas à rede. A Figura 1 ilustra a representação de uma política ACL (*Access Control List*) nas diferentes sintaxes dos firewalls Check Point, Palo Alto, Cisco PIX (*Private Internet eXchange*), Cisco ASA, IPTables e OpenFlow[2]. É importante ressaltar que firewalls como Check Point, Palo Alto e Cisco estão entre os mais utilizados no mercado (Gartner, 2021).

Figura 1 – Política ACL em diferentes sintaxes de firewalls

Check Point:
*mgmt_cli add access-rule layer "***inside***" position* **1** *name "***permit-http***" source "10.0.0.0/24"**
*destination "***any***" service "***HTTP***" action "***accept***"*

Palo Alto:
set rulebase security rule **permit-http** *from* **inside** *to* **outside** *source* **10.0.0.0 255.255.255.0**
destination **any** *service* **http** *action* **allow** *before* **all**

Cisco PIX:
access-list **1 permit tcp 10.0.0.0 255.255.255.0 any** *eq* **http**

Cisco ASA:
access-list **inside_access_in** *line* **1** *extended* **permit tcp 10.0.0.0 255.255.255.0 any** *eq* **http**

IPTables:
iptables -I FORWARD **1** *-s* **10.0.0.0/255.255.255.0** *-d* **0.0.0.0/0.0.0.0** *-p* **tcp** *--dport* **80** *-j* **ACCEPT**

OpenFlow:
ovs-ofctl add-flow ***br0*** *dl_type=0x800,priority=****65535****,nw_src=****10.0.0.0/255.255.255.0****,*
*nw_dst=****0.0.0.0/0.0.0.0****,nw_proto=****6****,tcp_dst=****80****,action=****normal***

Frequentemente, as linguagens de configuração podem variar para um mesmo fabricante. Cada fabricante pode oferecer diversas linhas de equipamentos, cada qual com uma linguagem de configuração específica. Por exemplo, o número de termos fixos (partes estáticas da sintaxe) para descrever uma mesma política do tipo ACL para um

[2]Nós utilizamos o OpenFlow como referência por ser o padrão de fato utilizado na indústria no que diz respeito a redes SDN.

mesmo fabricante como a Cisco, pode dobrar se precisarmos aplicá-la nas linhas PIX e ASA.

Além disso, as linguagens de configuração podem variar até mesmo para uma mesma linha de equipamentos, por causa do modelo ou versão. Por exemplo, considerando a linha Cisco ASA 5500, a sintaxe de representação de políticas pode mudar substancialmente de uma versão do software para outra (*e.g.*, da 8.2 para a 8.4), como ilustrada na Figura 2.

Figura 2 – Sintaxe NAT 1to1 versões Cisco ASA

Cisco ASA 8.2:
*static (***inside**,**outside***)* **tcp 200.19.0.30 90 10.0.0.30 80** *netmask* **255.255.255.255**
tcp **0 0** *udp* **0**

Cisco ASA 8.4:
object network **obj-10.0.0.30**
host **10.0.0.30**
nat static **200.19.0.30** *service* **tcp 80 90**

Diferentes fatores, como custos de aquisição ou a evolução dos equipamentos, podem provocar a instalação de firewalls de linhas ou fabricantes diversos em uma rede. Intuitivamente, quanto mais diversificadas as soluções de firewalls utilizadas, maior tende a ser o seu custo operacional. Os administradores precisam conhecer e traduzir regras para sintaxes específicas das soluções de firewall instaladas (MANSMANN; GöBEL; CHESWICK, 2012; AL-SHAER; HAMED, 2004). Consequentemente, a especificação e aplicação das regras em diferentes sintaxes potencializa erros de configuração que podem impactar a aplicação e a concretização da política de segurança na instituição (VORONKOV et al., 2017; TRAN; AL-SHAER; BOUTABA, 2007). De fato, estudos recentes apontam que erros de configuração são responsáveis por grande parte dos incidentes de segurança (IBM X-Force, 2020; Oracle, 2020), chegando a mais de 80% em alguns casos. Previsões recentes apontam que 99% das violações em firewalls serão causadas por configurações incorretas e não por falha das soluções até 2023 (Gartner, 2019).

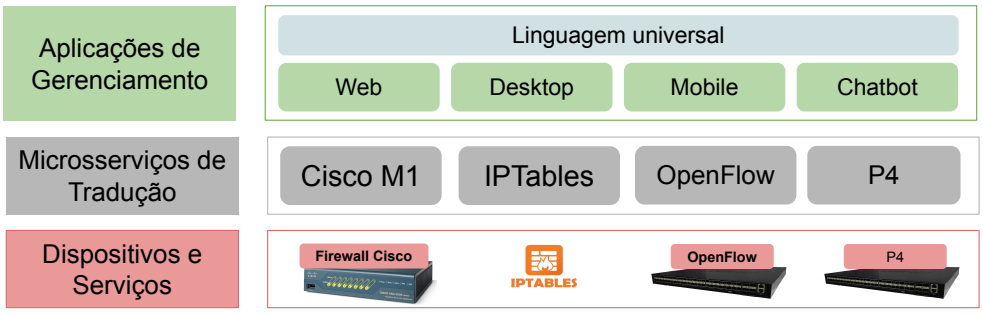
Levando em consideração o problema apresentado, o nosso objetivo é responder as seguintes perguntas: (a) É possível construírmos uma arquitetura de software para gerenciar equipamentos de firewall diversos em redes híbridas? (b) É possível representarmos políticas de firewalls modernos e de firewalls de novas gerações através de uma linguagem comum, intuitiva e compreensível por operadores de rede? (c) É possível

reduzirmos a complexidade do gerenciamento de equipamentos de firewall, oriunda frequentemente da complexidade das sintaxes e semânticas das linguagens específicas?

1.2 Proposta

Neste trabalho, propomos uma arquitetura em camadas, modular e extensível, para permitir o gerenciamento integrado de firewalls em redes híbridas. Uma visão geral da arquitetura proposta, denominada FWunify, é apresentada na Figura 3. A organização em camadas favorece a separação de conceitos (*e.g.*, aplicações de gestão e serviços de tradução) e estruturação da solução. Dentro de cada camada há módulos independentes entre si. Essa modularização permite alteração (adição, remoção ou manutenção) de recursos de maneira simples e rápida – sem impactar nos outros módulos e refletindo minimamente nas camadas adjacentes apenas. A inclusão de novos módulos nas camadas deve ser organizada de forma a não impactar as camadas adjacentes. Por exemplo, módulos de tradução de regras podem ser facilmente acrescentados na solução para atender o gerenciamento de novos tipos ou famílias de firewall. O módulo de tradução é concebido como um microsserviço fracamente acoplado, isto é, que pode ser desenvolvido de maneira independente e simplesmente adicionado à solução sem impactar em sua operação.

Figura 3 – Visão geral da arquitetura proposta resumida



Para a arquitetura ser adotada na prática é fundamental padronizar a linguagem de definição de políticas de segurança utilizadas em firewalls. Para atender essa demanda, propomos a linguagem FWlang, que é baseada no conceito de intenções, inspirada no recente advento das chamadas redes baseadas em intenções, ou IBN (*Intent-Based Networking*) (CLEMM et al., 2019; Zeydan; Turk, 2020; WEI; PENG; LIU, 2020). Como

é baseada em intenções, a FWlang permite a elaboração de regras em nível superior de abstração, pois define intenções ao invés de políticas. A FWlang é retrocompatível e pode substituir múltiplas linguagens específicas de soluções de firewall existentes. A FWlang é extensível e pode representar regras de firewalls modernos, como *traffic shaping*, filtros de URL (*Uniform Resource Locator*) ou regras de roteamento.

1.3 Contribuições

As principais contribuições deste trabalho são:

1. FWunify: uma arquitetura em camadas, modular e extensível, para orientar o desenvolvimento de soluções integradas de gerenciamento de firewalls. A arquitetura deve ser capaz de representar os requisitos necessários para representar, traduzir e aplicar as políticas nos equipamentos de firewall em redes híbridas (Seção 3).

2. FWlang: uma linguagem para definição de regras de firewall baseadas em intenções. A especificação de uma linguagem abstrata e genérica o suficiente para representar políticas de segurança utilizadas em firewalls modernos (Seção 4).

3. Implementação e disponibilização (código aberto e livre) de uma versão funcional da arquitetura de software FWunify, que utiliza a linguagem FWlang para definição de intenções de segurança. O sistema implementado contempla as sete camadas da arquitetura proposta e realiza a tradução e aplicação das políticas em firewalls Cisco (ASA 5520), IPTables e OpenFlow (baseado no Open vSwitch).

1.4 Estrutura da Dissertação

A dissertação está organizada da seguinte forma:

Capítulo 1 - Introdução do problema e esboço da nossa solução para o gerenciamento integrado de firewalls.

Capítulo 2 - Contextualização e discussão sobre soluções de gerenciamento de firewalls existentes, e das linguagens que possibilitam a representação de políticas de segurança.

Capítulo 3 - Apresentação da arquitetura de software proposta, que visa orientar o desenvolvimento de soluções de gerenciamento de firewalls em redes híbridas.

Capítulo 4 - Apresentação da linguagem de representação de políticas de segurança proposta.

Capítulo 5 - Detalhamento da implementação da arquitetura e da linguagem em um protótipo funcional que gerencia políticas de segurança em firewalls diversos.

Capítulo 6 - Apresentação e discussão dos resultados de avaliação da FWunify e FWlang.

Capítulo 7 - Considerações finais, resumindo os resultados, contribuições, trabalhos futuros e publicações.

2 ESTADO DA ARTE

Neste capítulo, apresentamos uma introdução e discussão sobre soluções de gerenciamento de firewalls (Seção 2.1) e linguagens de representação de políticas de segurança (Seção 2.2). Analisamos as soluções de gerenciamento de firewalls com relação ao tipo de arquitetura, a disponibilidade de código e o escopo de aplicação. Similarmente, agrupamos e discutimos as linguagens observando características como suporte a representação de diferentes tipos de políticas de segurança utilizadas em firewalls modernos, além da adoção de conceitos recentes, como intenções, para simplificar a representação de políticas.

2.1 Soluções de gerenciamento de firewalls

Soluções de gerenciamento de firewalls têm por objetivo principal traduzir e aplicar as políticas de segurança nos equipamentos propriamente ditos. Tipicamente, a tradução é realizada de linguagens de alto nível (*e.g*, interfaces gráficas ou linguagens estruturadas) para comandos de baixo nível, isto é, na sintaxe específica de cada firewall.

As soluções comerciais podem ser caracterizadas como um software específico, ou seja, projetado para o gerenciamento de um tipo ou uma família de firewalls de um fabricante, como é o caso das soluções proprietárias Cisco ASDM (*Adaptive Security Device Manager*) (Cisco Systems, 2018) e Fortinet FortiManager (Fortinet, 2021). Similarmente, também temos um conjunto de soluções livres, como Gufw (Gufw Project, 2021), pfSense (pfSense, 2020) e Mignis (Adão et al., 2014), projetadas para o gerenciamento de firewalls específicos (*e.g.*, baseados no filtro de pacotes IPTables).

No contexto de firewalls para redes que seguem o paradigma SDN, predominam as soluções específicas para padrões e protocolos como OpenFlow e P4. Tanto os firewalls baseados em OpenFlow (*e.g.*, Firewall for POX (OTHMAN et al., 2017), Fireflow (FIESSLER et al., 2018), REFLO (Visoottiviseth et al., 2017), FlowTracker (Vinh Tran; Ahn, 2016), Fortresss (CAPROLU; RAPONI; PIETRO, 2019), AI-SDNF (Cheng et al., 2018)), quanto os baseados em P4 (*e.g.*, SMPU-P4 (VÖRÖS; KISS, 2016), P4GUARD (DATTA et al., 2018)), podem ser caracterizados como soluções monolíticas, isto é, projetadas e pensadas para um determinado fim e cenário em particular. É importante destacar também que nenhuma das soluções para redes SDN possui referência à implementação ou código fonte livre e disponível. Isso pode ser

caracterizado como um desafio adicional no desenvolvimento e adoção de soluções de firewall baseadas em SDN, uma vez que esse paradigma promove conceitos como *open networking*, cujas iniciativas e ações são coordenadas pela *Open Networking Foundation* (ONF)[1], um consórcio de empresas globais de telecomunicações e grandes empresas da área de tecnologia.

A Tabela 1 resume três características de soluções de gerenciamento de firewalls, incluindo tipo de arquitetura de software, disponibilidade do código fonte e escopo de aplicação. Além das soluções existentes, incluímos também a FWunify, proposta neste trabalho.

Tabela 1 – Soluções de gerenciamento de firewalls

Solução	**Arquitetura**	**Código fonte**	**Escopo**
Cisco ASDM	Monolítica	Proprietário	Cisco
FortiManager	Monolítica	Proprietário	Fortinet
Gufw	Monolítica	Livre/Disponível	IPTables
Mignis	Monolítica	Livre/Disponível	IPTables
pfSense	Monolítica	Livre/Disponível	pfSense
Firewall for POX	Monolítica	Não disponibilizado	SDN (OpenFlow)
Fireflow	Monolítica	Não disponibilizado	SDN (OpenFlow)
REFLO	Monolítica	Não disponibilizado	SDN (OpenFlow)
FlowTracker	Monolítica	Não disponibilizado	SDN (OpenFlow)
Fortress	Monolítica	Não disponibilizado	SDN (OpenFlow)
AI-SDNF	Monolítica	Não disponibilizado	SDN (OpenFlow)
SMPU-P4	Monolítica	Não disponibilizado	SDN (P4)
P4GUARD	Monolítica	Não disponibilizado	SDN (P4)
FWunify	Em camadas e modular	Livre/Disponível	Todos

Como podemos observar, há dois conjuntos distintos de soluções em termos de arquitetura de software: as monolíticas, pensadas como uma aplicação para um determinado cenário, e as em camadas e modulares, resumindo-se à FWunify, para o gerenciamento de equipamentos de firewalls em ambientes de rede corporativos. A arquitetura em camadas e modular possui uma relação direta com o escopo, como pode ser observado na Tabela 1. Enquanto que a maioria das soluções é monolítica, o que dificulta substancialmente a incorporação e o gerenciamento de outros tipos de firewalls, a FWunify representa uma classe de soluções pensada e projetada para atender cenários como redes híbridas, onde a quantidade e a diversidade de firewalls é um fator importante

[1] <https://opennetworking.org>

a ser considerado. A FWunify permite, através de uma acoplação fraca entre as camadas e a independência entre os módulos dentro das camadas (ver detalhes na Seção 3), a adição e o gerenciamento de novas soluções de firewall. Por exemplo, no contexto de SDNs, na FWunify podemos incluir módulos de tradução para protocolos e padrões como POF ou ForCES.

2.2 Linguagens de representação de políticas

Existem essencialmente dois grupos de linguagens para representar políticas de segurança, como pode ser observado na Tabela 2. O primeiro grupo é composto pelas linguagens específicas de domínio, como RichLanguage (Firewalld, 2021), PDLz (Lobo; Marchi; Provetti, 2012), LAI (TIAN et al., 2019) e NILE (JACOBS et al., 2018). A RichLanguage e a PDLz são exemplos de linguagens criadas para representar regras de firewalls específicos, com os baseados no IPTables. A LAI é um caso interessante por ser uma linguagem projetada para gerenciar políticas ACLs em WANs (*Wide Area Networks*) privadas. Mais especificamente, a linguagem permite a adição, remoção e migração de regras de controle de tráfego nos diferentes roteadores que compõem uma WAN.

Tabela 2 – Linguagens para representação de políticas de segurança

	Políticas utilizadas em firewalls modernos							
Linguagem	**ACL**	**NAT 1to1**	**NAT Nto1**	**Traffic Shaping**	**URL Filter**	**Roteamento estático**	**Baseada em Intenções**	**Domínio de Aplicação**
RichLanguage	✓	✓	✓	✗	✗	✗	✗	IPTables
PDLz	✗	✗	✗	✗	✗	✓	✗	IPTables
LAI	✓	✗	✗	✗	✗	✗	✓	WAN
NILE	✓	✗	✗	✓	✗	✗	✓	NFVs
FLIP	✓	✗	✗	✗	✗	✗	✗	Qualquer
FWS	✓	✓	✓	✗	✗	✗	✗	Qualquer
FIRMATO	✓	✗	✗	✗	✗	✗	✗	Qualquer
AFPL2	✓	✓	✓	✗	✗	✗	✗	Qualquer
FWLang	✓	✓	✓	✓	✓	✓	✓	Qualquer

A linguagem NILE foi projetada para o gerenciar serviços no contexto de NFV (*Network Functions Virtualization*). Além de realizar o encadeamento de NFVs, a linguagem permite também a aplicação de políticas simples de ACL e *traffic shaping* no tráfego entre as funções virtualizadas. Neste primeiro grupo, como pode ser observado na Tabela 2, outras linguagens, como a RichLanguage, suportam também políticas de NAT (*Network Address Translation*) dos tipos 1to1 e Nto1.

O segundo grupo é formado pelas linguagens que propõem uma abstração mais geral, isto é, que pode ser utilizada para representar políticas de diversos tipos de firewalls. Este é o caso de linguagens como a FLIP (ZHANG et al., 2007), FIRMATO (BARTAL et al., 2004), FWS (BODEI et al., 2018), AFPL2 (POZO; VARELA-VACA; GASCA, 2009) e FWlang. Linguagens como FWS e AFPL2 contemplam a representação de políticas do tipo ACL, NAT 1to1 e NAT Nto1. Neste grupo de linguagens, é importante destacar que a única linguagem a suportar os seis tipos (ACL, NAT 1to1, NAT Nto1, *traffic shapping*, filtros de URL e roteamento estático - ver detalhes na Seção 4) de políticas de firewalls modernos é a FWlang.

Outra característica que merece destaque é o uso de intenções para representar políticas de firewalls. O desenvolvimento e a adoção das redes baseadas em intenções (IBNs) vem ganhando força e popularização devido ao apoio de grandes fabricantes de equipamentos de rede e segurança, como a Cisco (Cisco Systems, 2021), Juniper (COONEY, 2020), Huawei (Huawei Technologies, 2019) e VMware (VMware, 2020), e o aumento do número de pesquisas sobre o tema (Zeydan; Turk, 2020; WEI; PENG; LIU, 2020). Para simplificar a representação de políticas de rede ou segurança, linguagens como a LAI e NILE são baseadas na definição de intenções. Similarmente, seguindo essa tendência de mercado e pesquisa, e principalmente a premissa de simplificar a representação das políticas de segurança da rede, a FWlang também utiliza intenções. Resumidamente, com intenções, o operador não precisa mais aprender e conhecer a sintaxe e semântica específica das soluções de firewalls, potencializando a redução de custos de implantação e operação através de uma curva de aprendizagem menor e simplicidade e uniformidade na descrição de políticas de segurança.

3 FWUNIFY: ARQUITETURA DE SOFTWARE

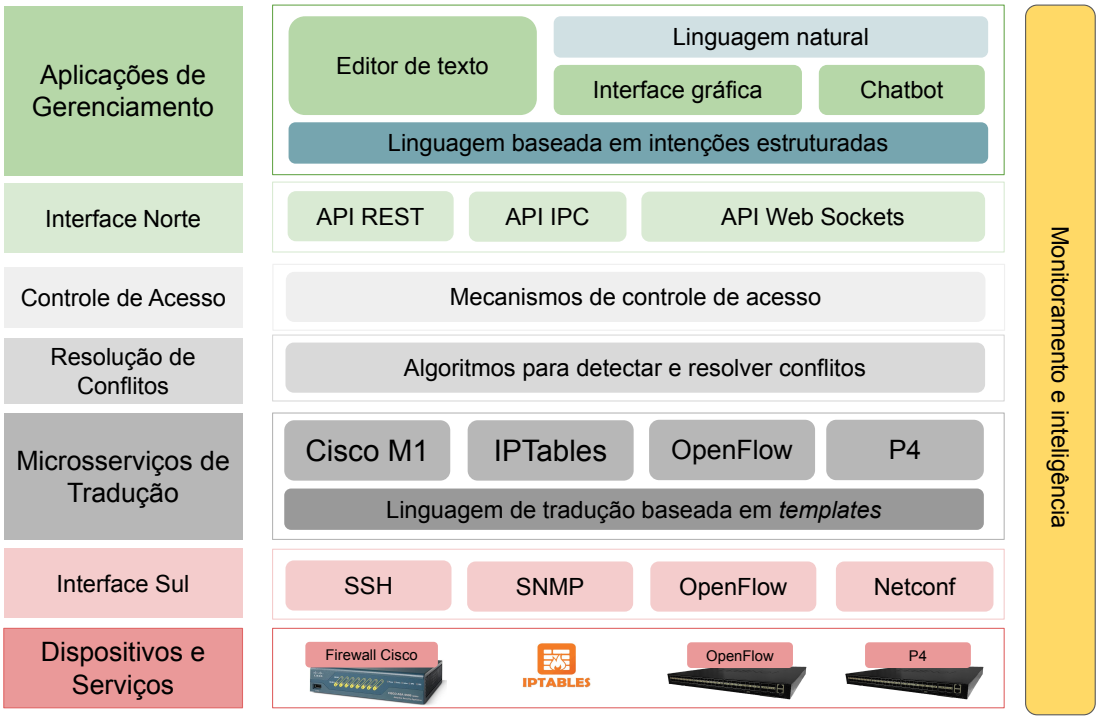
Neste capítulo apresentamos e discutimos a FWunify, uma arquitetura de software que tem como principal objetivo balizar o desenvolvimento de soluções de gerenciamento de firewalls em redes híbridas. A FWunify permite que a partir da representação genérica de uma política, sejam gerados os comandos que atendam a necessidade expressa a nível de rede. A arquitetura é flexível e modular para suportar os mais diferentes fabricantes, linhas e modelos de soluções de firewall disponíveis para redes híbridas, baseadas em tecnologias e protocolos como ASA 5500-X (Cisco), IPTables, pfSense, OpenFlow e P4.

A arquitetura FWunify é organizada em sete camadas, como mostra a Figura 4. As colunas da esquerda e da direita contém, respectivamente, as camadas e exemplos de módulos para tecnologias viabilizadoras da respectiva camada. A arquitetura em camadas permite que módulos sejam adicionados ou removidos conforme a necessidade da rede gerenciada, impactando apenas e minimamente nas camadas adjacentes. Por exemplo, um novo módulo, que gere políticas para um determinado tipo de firewall, pode ser adicionado à camada de microsserviços de tradução. Para isso, as camadas superiores (*e.g*, uma API (*Application Programming Interface*) REST (*Representational State Transfer*)) precisam ser informadas do novo módulo, para que enviem as intenções para a tradução. Dependendo do método utilizado para aplicar as políticas no firewall, esse novo microsserviço de tradução poderá: (a) utilizar um dos módulos disponíveis na interface sul, ou (b) demandar a criação ou instalação de um módulo que suporte a forma de acesso do novo firewall. A seguir, cada camada da arquitetura é detalhada em ordem de cima para baixo.

Na camada superior, *Aplicações de Gerenciamento*, os módulos representam as interfaces (*i.e.*, pontos de interação com o usuário final) utilizadas para a definição, controle e ativação das políticas de segurança na rede. As interfaces podem ser implementadas utilizando diferentes recursos e tecnologias. A alternativa mais simples pode ser considerada um *editor de texto*, utilizado para a edição das intenções de segurança descritas em uma linguagem baseada em intenções estruturadas. A partir de uma ferramenta simples, como um comando `curl`[1], os operadores da rede podem facilmente enviar a intenção para tradução através de uma API REST, por exemplo. Outras opções, como *interfaces gráficas* ou *chatbots*, podem interpretar intenções descritas em linguagem natural (*e.g*, “Eu quero bloquear o tráfego HTTP da rede do

[1]<https://curl.se>

Figura 4 – Arquitetura proposta detalhada



laboratório de informática"), oferecendo um nível ainda maior de abstração ao operador da rede. É importante ressaltar que as soluções de mais alto nível, como as que utilizam linguagem natural, podem utilizar a linguagem baseada em intenções estruturadas para facilitar o processo de tradução da demanda do usuário, isto é, a demanda é traduzida de linguagem natural para uma linguagem estruturada, que é verificável e processada com maior acurácia nas camadas inferiores da arquitetura.

É importante ressaltar que grande parte dos operadores de redes preferem alternativas simples, como interfaces de linha de comando, para diversos tipos de atividades relacionadas ao gerenciamento de firewalls (BOTTA et al., 2007; HABER; BAILEY, 2007; VORONKOV; MARTUCCI; LINDSKOG, 2019). Por outro lado, soluções como interfaces gráficas e *chatbots* são também desejáveis na perspectiva de administradores de sistemas, em especial aqueles que gastam quatro ou menos horas semanais gerenciando firewalls (VORONKOV; MARTUCCI; LINDSKOG, 2019). Uma interface gráfica pode facilitar a criação, organização e visualização das intenções de segurança para os administradores de sistemas. Similarmente, um *chatbot*, utilizando intenções em linguagem natural (*e.g.*, "Quero bloquear todo o tráfego FTP (*File Transfer Protocol*) da rede."), pode simplificar e agilizar a gestão de políticas de segurança

para gestores da informação. Entretanto, para simplificar, padronizar e agilizar o desenvolvimento de aplicações de gerenciamento, é importante que as interfaces utilizem uma mesma *linguagem universal*, como discutido no Capítulo 4).

A camada *Interface Norte* é responsável pela comunicação entre as aplicações de gerenciamento e a camada de tradutores. A Interface Norte pode ser implementada como uma API do tipo REST, que é utilizada para receber e enviar a descrição das políticas de segurança entre as camadas superior e inferior da arquitetura.

A camada *Controle de Acesso* é responsável pela autenticação, autorização e responsabilização (*accountability*) dos usuários. Mecanismos de autenticação, como *login* e senha, garantem que um determinado usuário corresponde a quem ele diz ser, além restringirem ou liberarem o acesso do usuário aos recursos dependendo de seu nível de permissão. Soluções comerciais como as da Cisco (Cisco Systems, 2019) e Palo Alto (Palo Alto Networks, 2021) utilizam papéis e diferentes níveis de permissão para controlar o acesso dos usuários aos recursos dos firewalls. Por exemplo, a Cisco define quinze níveis de acesso que podem ser personalizados para permitir determinados tipos de políticas. A camada *Controle de Acesso* foi pensada para abrigar mecanismos de controle de acesso deste tipo, isto é, que permitam além da autenticação, a definição de níveis de controle de acesso baseado em papéis e ações.

O objetivo da camada de controle de acesso é oferecer níveis de privilégios personalizáveis, que possibilitem a criação de perfis de operadores com diferentes permissões, diferentes escopos de ação ou ainda diferentes níveis de granularidade no contexto de um escopo específico (e.g., apenas adicionar intenções ACL em um laboratório de informática). Como exemplo de utilização baseada em escopo, podemos considerar uma instituição de ensino multi-campi, onde o gerente de segurança possui o nível de permissão máximo, podendo aplicar qualquer tipo de configuração em todos os firewalls da instituição. Um técnico de TI da instituição pode ter o escopo limitado a intenções dos tipos ACL e *traffic shaping*) e que dizem respeito exclusivamente a sua unidade, por exemplo. Um operador de suporte da instituição, responsável pelo primeiro nível de atendimento de incidentes de segurança, pode ter o escopo limitado a *hosts* da instituição, políticas do tipo ACL e ação do tipo adicionar. Em outras palavras, no primeiro sinal de um incidente de segurança, o operador de suporte podem imediatamente adicionar uma ACL para negar o tráfego de ou para algum *host* em particular. Entretanto, apenas um operador de mais alto nível, como o gerente de segurança, poderá remover a ACL após o incidente de segurança ter sido avaliado, devidamente investigado e resolvido.

O objetivo da camada de *Resolução de Conflitos* é realizar a verificação das intenções de segurança definidas pelo usuário. Existem essencialmente duas abordagens para detecção e resolução de conflitos em firewalls. A primeira, representada por soluções como PreFirewall (MORZHOV; ALEKSEEV; NIKITINSKIY, 2016) e FAME (HU; AHN; KULKARNI, 2012), realizam a verificação das novas políticas adicionadas ao firewall para evitar colisões, redundâncias e problemas de sobreposição em regras. A segunda, utilizada por soluções como FlowGuard (HU et al., 2014) e FlowMon (HU et al., 2019) no contexto de SDNs, que monitoram os fluxos e caminhos de dados na rede em busca de tráfegos que indiquem violações nas políticas de segurança em uso. Soluções utilizadas na camada de resolução de conflitos podem, por exemplo, evitar a inserção de políticas conflitantes que liberem um tráfego que deveria ser bloqueado segundo a política de segurança de uma instituição.

Os *Microsserviços de Tradução* recebem as intenções de segurança através da interface norte e as traduzem para comandos específicos a cada tipo de solução de firewall. Para facilitar a tradução, os microsserviços são construídos utilizando uma linguagem de tradução baseadas em *templates*, que permite a rápida composição das políticas de segurança na sintaxe específica de cada firewall. Na prática, o microsserviço Cisco M1 irá traduzir as intenções de segurança para comandos específico de um firewall ASA 5500-X, cuja versão do *firmware* é 8.2 ou 8.4[2], por exemplo. Tanto no caso de *firmwares* com múltiplas versões de sintaxe quanto no caso de padrões como o OpenFlow, pode haver um serviço de tradução para cada versão específica da solução ou do protocolo (*e.g.*, v1.0, v1.1, v1.2, v1.3, v1.4, v1.5). Supondo que uma rede de grande porte possui sete soluções diferentes de firewall, serão necessários até sete microsserviços de tradução, um para cada linha distinta de firewalls.

A *interface sul* é responsável pela ligação entre as camadas de tradução e de dispositivos e serviços. As regras geradas pelos microsserviços de tradução são enviadas para as diferentes soluções de firewall utilizando os respectivos protocolos. Por exemplo, um firewall Linux (IPTables) ou Cisco (ASA 5500-X) pode ser configurado utilizando o protocolo SSH (*Secure Socket Shell*). Já um firewall SDN pode ser configurado utilizando o próprio padrão do OpenFlow (*e.g.*, conexão TLS (*Transport Layer Security*)), por exemplo.

Os *dispositivos e serviços* correspondem às soluções de firewall utilizadas na rede. Para cada nova solução adicionada na rede, será necessário associar um microsserviço

[2]Conforme discutido na Seção 1.1, podem haver diferenças significativas de sintaxe de uma versão para outra do *firmware*.

de tradução e um protocolo da interface sul que possibilitem a tradução e aplicação das políticas.

Por fim, temos a camada transversal de *monitoramento e inteligência*, composta por recursos como mecanismos de automação, telemetria e aprendizagem de máquina para dar suporte às demais camadas da arquitetura. Por exemplo, dados de monitoramento podem suportar o operador na composição de uma política a ser aplicada. Em outras palavras, esta camada transversal pode permitir a retro-alimentação automática da solução, indo de encontro à construção de uma solução autônoma de gerenciamento de firewalls. Por exemplo, na prática, quando o sistema identificar tráfego suspeito na rede, políticas de segurança específicas para este tráfego podem ser definidas e aplicadas nos firewalls pelo próprio sistema, de forma automatizada.

4 FWLANG: LINGUAGEM DE REPRESENTAÇÃO DE POLÍTICAS

Para superar os desafios relacionados ao gerenciamento de redes heterogêneas (ver Seção 1.1), o conceito de redes baseadas em intenções, ou IBN, tem sido explorado recentemente pela indústria (Cisco Systems, 2017; VMware, 2020; Huawei Technologies, 2019) e pela academia (SINGH; AUJLA; BALI, 2020; GAO; CONTRERAS; RANDRIAMASY, 2020; WEI; PENG; LIU, 2020; NAGENDRA et al., 2020; HEORHIADI et al., 2018; SANVITO et al., 2018). A intenção ("*intent*") pode ser considerada uma evolução do conceito de política ("*policy*") (Zeydan; Turk, 2020). Em síntese, uma intenção é uma representação mais abstrata que uma política de rede. O objetivo é fornecer uma camada de abstração que facilite o gerenciamento de dispositivos através da automatização de procedimentos que seriam realizados manualmente.

Em consonância com os avanços recentes no contexto de gerenciamento de redes baseadas em intenções, propomos a **FWlang**: uma linguagem estruturada de representação de intenções, voltada a ampliar a manutenibilidade de firewalls em redes híbridas. Através da FWlang, usuários com os mais variados graus de instrução – desde pouco conhecimento técnico específico até ampla experiência com administração de firewalls – devem ser capazes de definir e aplicar políticas de segurança, sem precisar acessar e conhecer as sintaxes e especificidades de quaisquer tipos de firewalls.

Como um primeiro passo para definir a FWlang, elencamos um conjunto inicial de políticas, consideradas fundamentais. Para escolher esse conjunto realizamos um estudo englobando: (a) políticas implementadas em quatro ambientes reais, (b) documentação de fabricantes de equipamentos de segurança, e (c) literatura científica. Os ambientes reais[1] são compostos por dois firewalls de borda responsáveis pela segurança de redes de várias centenas de máquinas, um firewall interno que protege uma rede de filiais da instituição, e um firewall de um servidor Web que é executado em um data center terceirizado. As documentações oficiais consultadas incluem materiais publicados pelas fabricantes Cisco (Cisco Systems, 2020), Palo Alto (Palo Alto Networks, 2020) e Check Point (Check Point, 2020). Em relação à literatura científica, analisamos trabalhos que abordam políticas de segurança utilizadas em firewalls (SCARFONE; HOFFMAN, 2009; KRIT; HAIMOUD, 2017; BODEI et al., 2018).

A Tabela 3 apresenta um resumo das principais políticas utilizadas por firewalls modernos. Como podemos observar, soluções de mercado consolidadas, como Cisco,

[1] Os detalhes são omitidos para preservar as instituições e empresas.

Palo Alto, Fortinet e Check Point, possuem em comum as políticas dos tipos ACL, filtro de URL, *traffic shaping*, roteamento estático, NAT 1to1 e NAT Nto1. Algumas soluções, como a Palo Alto, possuem também recursos para expressar políticas mais específicas, como *Decryption*[2] e *DoS Protection*[3]. Como essas políticas não são comuns às diferentes soluções de firewall, elas foram excluídas do conjunto apresentado na tabela.

Tabela 3 – Políticas suportadas por firewalls modernos

Solução	ACL	Filtro de URL	Traffic Shaping	Roteamento estático	NAT 1to1	NAT Nto1
Cisco NGFW	✓	✓	✓	✓	✓	✓
Palo Alto NGFW	✓	✓	✓	✓	✓	✓
Fortinet NGFW	✓	✓	✓	✓	✓	✓
Check Point NGFW	✓	✓	✓	✓	✓	✓
PFSense	✓		✓	✓	✓	✓
IPTables, Mignis	✓				✓	✓
FlowTracker, P4GUARD	✓					

Soluções livres, como o pfSense (pfSense, 2020), suportam cinco das seis regras suportadas por firewalls comerciais. A política encontrada com menor frequência em soluções livres é a de filtro de URL, que é tipicamente atribuída a outras soluções livres da camada de aplicação, como o Squid[4]. A maioria das demais soluções, como o IPTables (Netfilter, 2021), Mignis (Adão et al., 2014), SDN/Fireflow (FIESSLER et al., 2018), SDN/FlowTracker (Vinh Tran; Ahn, 2016) e SDN/P4GUARD (DATTA et al., 2018) limitam-se a políticas dos tipos ACL e NAT.

Como resultado do estudo, chegamos a seis políticas de firewall para especificar a primeira versão da FWlang, resumidas na Tabela 4. Por exemplo, uma política do tipo ACL tem seus requisitos divididos em dois grupos. O primeiro representa os termos que são obrigatórios (destacados em itálico na tabela) para descrição da política, como origem e destino dos pacotes, tipo de tráfego a ser tratado, a ação a ser tomada (bloquear/permitir), e a ordem de prioridade de execução da regra no firewall. Requisitos como ativação do serviço de registro de eventos (*logs*), definição do intervalo de tempo que a regra ficará ativa e descrição são campos opcionais oferecidos por algumas das soluções de firewall analisadas. Os termos obrigatórios para composição das políticas são apresentados a seguir.

Access Lists (ACLs) são utilizadas para regular (liberar/permitir) o tráfego entre redes distintas. As ACLs são tipicamente compostas pela origem e destino dos pacotes, o tipo de tráfego, a ação a ser realizada, além da ordem de prioridade da regra no firewall.

[2] <https://tinyurl.com/palo-alto-decryption>
[3] <https://tinyurl.com/palo-alto-dos>
[4] <http://www.squid-cache.org>

Tabela 4 – Políticas de firewalls modernos

Política	ACL	Filtro de URL	Traffic Shaping	Roteamento estático	NAT 1to1	NAT Nto1
Requisitos	*Origem* *Destino* *Tráfego* *Ação* *Prioridade* Serviço de logs Intervalo de tempo Descrição	*Origem* *Destino* *Tráfego* *Ação/Largura de banda* *Prioridade* Serviço de logs Intervalo de tempo Descrição	*Origem* *Destino* *Tráfego* *Largura de banda* *Prioridade* Serviço de logs Intervalo de tempo Descrição	*Origem/Destino* *Gateway* Intervalo de tempo Descrição	*Origem* *Destino* *Tráfego* Serviço de logs Descrição	*Origem* *Destino* Serviço de logs Descrição

Network Address Translation (NAT) 1to1 é um tipo de política utilizada para mapear o tráfego de um endereço IP público para um endereço IP privado. Uma regra NAT é composta pela origem e destino dos pacotes e pelo tipo de tráfego (*e.g.*, todo tipo de tráfego ou protocolo/portas específicas).

Network Address Translation (NAT) Nto1 é uma política bastante comum em firewalls de borda. Tipicamente, uma regra NAT Nto1 permite que vários endereços IP de uma rede interna tenham acesso à Internet através de um único IP público. Este tipo de política é frequentemente utilizado em redes IPv4 devido a limitação no número de endereços IPv4 públicos disponíveis para as instituições. A regra é composta pela origem (rede interna) e o destino (IP público).

Traffic Shaping é um tipo de regra utilizada para restringir a vazão de tráfego entre endereços IP de *hosts*, redes ou categorias (*e.g.*, redes sociais). A limitação da vazão pode ser aplicada a todo o tráfego ou a fluxos de dados específicos, como portas e protocolos. Para isso, uma política de *traffic shaping* requer a especificação da origem e do destino (*e.g.*, IP, rede, categoria), o tipo de tráfego afetado, a largura de banda estabelecida e a ordem de prioridade de processamento da regra no firewall.

Roteamento Estático é comum em firewalls que definem rotas entre redes. Nesses casos, o administrador do sistema define estaticamente o caminho dos pacotes para chegar até uma rede remota. As regras de roteamento estático são compostas pelos endereços IP (ou de rede) de origem ou destino e o *gateway* que deve ser utilizada para encaminhar os pacotes até a rede de destino.

Filtro de URL é um recurso de firewalls modernos que torna possível a filtragem dos pacotes com base na URL do site de destino. Além da URL, firewalls de nova geração também realizam a filtragem de tráfego baseado em categorias (*e.g.*, p2p, URLs de *phishing*). Estas políticas permitem o bloqueio ou a limitação de banda para conexões filtradas. Uma regra de filtro de URL é composta pela origem, destino (que pode ser uma categoria de tráfego) dos pacotes, a ação a ser tomada (permitir, bloquear, ou aplicar um limite de vazão no tráfego) e a ordem de prioridade de processamento da regra no firewall.

Para representar as seis políticas de segurança utilizando intenções, consideramos as linguagens NILE (JACOBS et al., 2018) e LAI (TIAN et al., 2019) como ponto de partida para a gramática da FWlang. Enquanto a NILE foi concebida para ser uma representação intermediária entre a descrição de uma intenção em linguagem natural e os comandos a serem aplicados na rede para o gerenciamento de NFVs, a LAI atende os requisitos de uma solução específica para o gerenciamento de ACLs em roteadores de uma WAN privada.

A linguagem LAI possui um conjunto limitado de marcadores, utilizados para definição de políticas ACL simples (*i.e*, com origem, destino e ação). Além disso, a linguagem incorpora essencialmente termos e sintaxe específica de ACLs de roteadores (*e.g*, *scope*, *isolate*, *fix*). Devido a essas peculiaridades, a utilização e extensão da LAI para representação das políticas de firewalls demandaria uma reformulação significativa (quase que total) da linguagem. Diferentemente, a NILE foi projetada para atender requisitos comuns de uma rede, como políticas de ACL e *traffic shaping* em dispositivos de rede que implementam e executam as funções virtuais da rede. Em síntese, a NILE utiliza termos intuitivos, corriqueiros e simples (*e.g.*, *from*, *to*, *traffic*, *throughput*) para administradores de sistemas expressarem políticas utilizadas em redes corporativas, sendo este um dos principais motivos da sua escolha da linguagem como ponto de partida para a especificação e implementação da FWlang. Um segundo fator que também contribui para a sua escolha foi o maior nível de abstração oferecido pela linguagem, apresentando marcadores que tornam a intenção facilmente legível e compreensível pelos operadores da rede.

A gramática da FWlang, em notação EBNF (*extended Backus–Naur form*) (JTC, 1996), é apresentada na Figura 5. O detalhamento dos termos da gramática pode ser visto no Apêndice A. A FWlang adota operadores básicos da NILE, como *from* e *to*, para definir origem e destino, e *start* e *end*, para definir o período em que uma intenção estará ativa. Para representar políticas de segurança de firewalls modernos, a FWlang estende a NILE adicionando marcadores às regras do tipo ACL e *traffic shaping*, bem como novos tipos de regras, isto é, NAT 1to1, NAT Nto1, roteamento estático e filtro de URL. Além disso, na FWlang, as intenções de segurança exigem operadores adicionais para expressar requisitos como prioridade, categorias de tráfegos, *gateways* ou controle do serviço de *logs*. É importante ressaltar que a gramática proposta pode ser estendida para incorporar novos tipos de políticas de segurança ou novos marcadores para atender requisitos de firewalls futuros.

Figura 5 – Gramática FWlang no formato EBNF

```
<intent>          ::= 'define intent' intent_name ':' <commands>
<commands>        ::= <command> {'\n' <command> }
<command>         ::= (<name> | <locations> | <rules> | <targets>
                  | <qos> | <middleboxes> | <order>) + [ <optional> ]
<name>            ::= 'name' <text>
<locations>       ::= 'from' <object> 'to' <object>
<object>          ::= 'endpoint(value | any)' | 'range(value)'
                  | 'category(value | any)'  | 'gateway(value)' | 'any'
<rules>           ::= (allow | deny) <traffic>
<targets>         ::= 'for' <traffic>
<qos>             ::= 'with' <metrics>
<metrics>         ::= 'throughput(value)'
<middleboxes>     ::= (add | del) <middlebox>
<middlebox>       ::= middlebox(mid_id {(',') mid_id}) | firewall(fw_id {(',') fw_id})
<order>           ::= 'order' <position>
<position>        ::= 'before(rule_name | all-intents)' | 'after(rule_name | all-intents)'
<traffic>         ::= 'traffic(value | any)' | 'port([<tuple>])' | 'any'
<tuple>           ::= 'protocol:' version ; 'src_port:' number ; 'dst_port:' number
<optional>        ::= <interval> | <log> | <description>
<interval>        ::= 'start' <data_time> '\n' 'end' <data_time>
<data_time>       ::= 'datetime(value)' | 'date(value)  | 'hour(value)'
<log>             ::= 'logging' <options>
<options>         ::= (enable | disable)
<description>     ::= 'description' <text>
<text>            ::= 'text(rule_name | rule_description)'
```

Os marcadores utilizados pela gramática foram definidos em um processo de três etapas: (a) análise dos marcadores originais da linguagem NILE; (b) análise dos termos tipicamente utilizados em firewalls; e (c) avaliação participativa com um grupo de potenciais usuários (operadores de sistemas) da linguagem. A primeira análise buscou identificar os termos presentes na linguagem NILE que poderiam ser reutilizados para a definição de intenções utilizadas em firewalls. Este foi o caso de marcadores como *from/to*, *start/end* e *add*, que possibilitam determinar requisitos mapeados das políticas, como origem e destino, intervalo de tempo e a indicação de que a intenção deve ser adicionada ao firewall. Na segunda etapa, avaliamos a terminologia das sintaxes específicas dos firewalls, como pode ser visto na Tabela 5. Essa análise resultou na incorporação de marcadores como *log* e *description* a gramática da FWlang.

O conjunto de marcadores definidos nas duas etapas anteriores foi submetido a uma avaliação participativa (MULLER; HASLWANTER; DAYTON, 1997; MELO et al., 2020). A avaliação foi realizada com um grupo de cinco administradores de sistemas com conhecimento prévio no gerenciamento de políticas de segurança. É importante ressaltar

Tabela 5 – Termos utilizados nas sintaxes dos firewalls

Termo/Firewall	Cisco	IPTables	OpenFlow	Palo Alto	Check Point
Origem	Definido pela posição	-s	nw_src=	source	source
Destino	Definido pela posição	-d	nw_dst=	destination	destination
Prioridade	line	-I	priority=	before/after	position
Ação	permit/deny	accept/drop	normal/drop	allow/deny	accept/drop
Tráfego	eq	-p –dport	nw_proto=, tcp_dst=	service	service
Largura de banda	Definido pela posição		ingress_policy_rate=	Egress max	Bandwidth
Serviço de logs	log	log	monitor	log-start	log
Intervalo de tempo	time-range			schedule	time
Descrição	remark	#		description	description

que estudos e especialistas apontam que cinco usuários, que entendem do domínio do problema, é um número estatístico bom para uma avaliação participativa (NIELSEN, 2018). Durante o processo, os participantes foram convidados a definir intenções utilizando a FWlang e expressar suas opiniões sobre os termos utilizados na representação das políticas. Ao final da atividade, uma discussão com todos os participantes foi realizada para analisar as sugestões propostas e verificar como elas poderiam ser incorporadas à linguagem. Este processo resultou na adoção de marcadores como *any* para representar qualquer origem, destino ou tráfego nas intenções e a substituição do marcador *block* (marcador original da linguagem NILE) por *deny*. O marcador *deny*, utilizado por diversos firewalls modernos como Cisco e Palo Alto (ver Tabela 5), foi considerado mais objetivo e intuitivo para indicar a ação de negar em políticas de ACL e filtros de URL. Todas as informações relativas a avaliação estão disponíveis online[5].

A Figura 6 ilustra quatro exemplos de intenções de segurança escritas com a FWlang. Como podemos observar, os firewalls estão identificados pelo termo *firewall* (*e.g.*, *firewall(openflow-1,iptables-1)*) e a operação de adicionar (*add*). Além do termo *firewall*, os firewall podem ser identificados pelo termo *middlebox* (e.g., "add *middlebox(cisco-1,iptables-1)"*). Apesar de o termo *firewall* ser potencialmente mais familiar aos administradores de sistemas, o termo *middlebox* foi mantido na linguagem por ser mais geral, isto é, representa quaisquer tipos de *appliances* de redes, como firewalls, NIDS (*Network Intrusion Detection System*), IDS (*Intrusion Detection System*), IPS (*Intrusion Prevention System*), produtos específicos para inspeção de tráfego SSL (*Secure Sockets Layer*)/TLS, entre outros. Apesar de o foco da FWunify e FWlang ser firewalls, nada impede de a solução e linguagem serem estendidas para outros tipos de *appliances* no futuro. Vale ressaltar também que os operadores de sistema, que integraram a avaliação participativa, concordaram com essa incorporação dos dois

[5] <https://tinyurl.com/fwlang-avaliacao-participativa>

marcadores equivalentes. Segundo eles, o termo *middlebox* não gera confusão e será útil em cenários futuros da linguagem.

O primeiro exemplo (Figura 6(a)) descreve uma política do tipo ACL identificada como "deny-netA-h100-http", que bloqueia o tráfego HTTP (*Hypertext Transfer Protocol*) (*deny traffic(http)*) da rede de origem 10.0.0.0/24 para o IP 200.19.0.100 (*to endpoint(200.19.0.100)*). É importante observar que a regra será adicionada (*add*) aos firewalls indicados no marcador *firewall* antes de todas as demais regras existentes (*order before(all-intents)*).

Figura 6 – Exemplos de intenções de segurança com FWlang

```
define intent acl:
name        text(deny-netA-h100-http)
from        range(10.0.0.0/24)
to          endpoint(200.19.0.100)
deny        traffic(http)
order       before(all-intents)
add         firewall(cisco-1,iptables-1,openflow-1)
```

(a) ACL

```
define intent traffic_shaping:
name        text(limit-net-h100-100m)
from        range(10.0.0.0/24)
to          endpoint(200.19.0.100)
order       before(all-intents)
for         traffic(udp/5555)
with        throughput(100Mbps)
add         firewall(openflow-1)
```

(b) *Traffic Shaping*

```
define intent url_filter:
name        text(filter-social-media)
from        range(10.0.0.0/24)
to          category(social-media)
deny        traffic(any)
order       after(filter-phishing)
log         enable
add         firewall(checkpoint-1)
```

(c) Filtro de URL

```
define intent route:
name        text(route-netA-gw1)
from        gateway(172.30.0.1)
to          range(10.0.20.0/24)
add         firewall(openflow-1)
```

(d) Roteamento Estático

O segundo exemplo, apresentado na Figura 6(b), define uma política do tipo *traffic shaping*, que limita a largura de banda na porta 5555/UDP (*User Datagram Protocol*) (*for traffic(udp/5555)*) da rede 10.0.0./24, para o IP 200.19.0.100, a 100 Mbps (*with throughput(100Mbps)*). Nesse caso, a ordem indica que a nova regra será adicionada antes de todas as outras políticas (*order before(all-intents)*) do firewall "openflow-1".

O terceiro exemplo, detalhado na Figura 6(c), é um filtro de URL que determina que todo tráfego gerado pela rede 10.0.0.0/24, passando pelo firewall "checkpoint-1", com destino para a categoria de tráfego "social-media", deve ser bloqueado (*deny traffic(any)*).

Essa política deve ser adicionada após a regra pré-existente “filter-phishing” e gerar registros de operação (*log enable*).

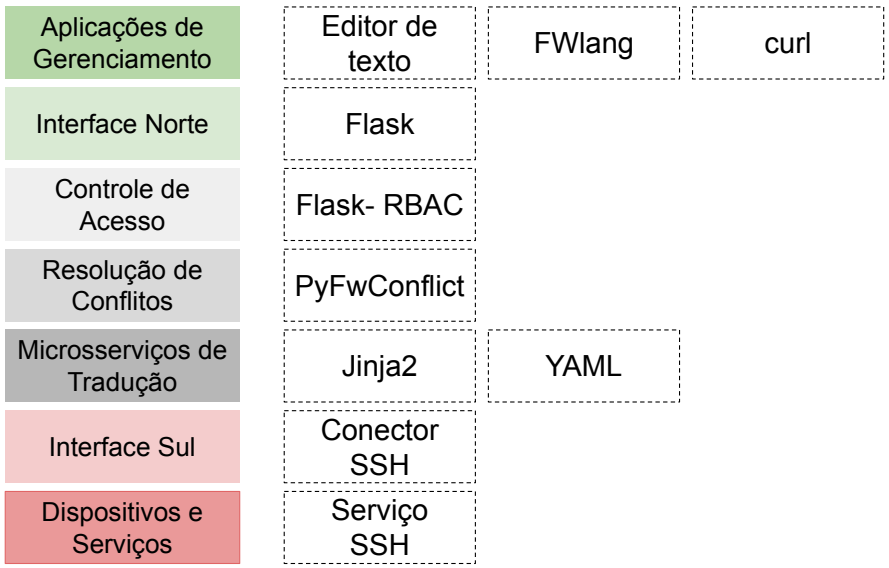
Por fim, o exemplo da Figura 6(d) representa uma política de roteamento. A intenção define que todo o tráfego destinado à rede 10.0.20.0/24 deve utilizar o *gateway* de saída representado pelo IP 172.30.0.1. Esta intenção será aplicada apenas no firewall “openflow-1”.

5 IMPLEMENTAÇÃO

Uma implementação funcional da arquitetura de software FWunify, incluindo a concretização da linguagem FWlang para a definição de políticas, está disponível em <https://github.com/mmfiorenza/fwunify>. A solução foi desenvolvida utilizando a linguagem de programação Python versão 3.7, e conta atualmente com aproximadamente 2500 linhas de código. Além de instanciar a arquitetura de software FWunify, a implementação atende também o conceito de intenções para o gerenciamento de redes, isto é, segue os princípios e o fluxo essencial de (a) submissão e (b) tradução das intenções e (c) aplicação das configurações (regras geradas) nos dispositivos da rede, conforme especificado no padrão em proposição no IETF (*Internet Engineering Task Force*) (SUN; LIU; XIE, 2019).

A versão atual da FWunify contempla uma API REST (interface norte), quatro microsserviços de tradução para firewalls Cisco ASA 5505, GNU/Linux IPTables, Open vSwitch e Palo Alto, além de dois conectores SSH (interface sul), um específico para Cisco e outro para equipamentos Linux. Na camada de dispositivos e serviços foram utilizados firewalls Cisco ASA 5520, Linux IPTables e OpenFlow, este último através do Open vSwitch. O microsserviço de tradução para Palo Alto é utilizado apenas para gerar e validar políticas do tipo filtro de URL. A Figura 7 resume as tecnologias utilizadas para viabilizar cada camada da arquitetura no FWunify.

Figura 7 – Tecnologias viabilizadoras de cada camada



Na camada aplicações de gerenciamento é utilizado um editor de texto para definir as intenções em arquivos, seguindo a sintaxe da FWlang (vide exemplos da Figura 6). Os arquivos das intenções são enviados para tradução, através da interface norte, utilizando o comando *curl* (<https://curl.se/>) via interface de linha de comando (CLI - *Command-line Interface*). Além dos arquivos das intenções, o administrador do sistema precisa informar também as suas credenciais de acesso para autenticação, autorização e registro de atividades. É importante ressaltar que CLI é um dos métodos mais utilizados por administradores de sistemas (BOTTA et al., 2007; HABER; BAILEY, 2007; VORONKOV; MARTUCCI; LINDSKOG, 2019; WONG, 2008).

A interface norte implementa uma API e um serviço REST utilizando framework Flask[1]. O envio das intenções é realizada através do método POST e do endereço "/" da API. Para o envio da intenção são necessárias: (a) as informações de autenticação do operador da rede (usuário e senha); e (b) a intenção descrita na sintaxe da FWlang, que deve ser enviada em formato binário (*e.g*, no caso do *curl* é utilizado o parâmetro "*–data-binary*"). O serviço REST recebe e processa a intenção. Na sequência, envia os dados da intenção para os microsserviços. Finalizada a tradução e aplicação da intenção, os microsserviços retornam uma resposta ao serviço REST, indicando o sucesso (código de status 200) ou a falha no processo. O serviço REST é responsável pela verificação e validação da sintaxe e dos valores atribuídos aos marcadores da FWlang. Como exemplos de valores de marcadores podemos citar endereços IP, nomes de *hosts*, definição de sub-redes (*e.g*, 10.0.0.0/23) no padrão CIDR (*Classless Inter-Domain Routing*) e definição de largura de banda em "Mbps" para políticas de *traffic shaping* ou filtros de URL. Este serviço REST também extrai os dados das intenções (*e.g.*, origem, destino, porta, protocolo e ações) necessários para compor a sintaxe nos tradutores especializados. Esses informações são adicionados em um dicionário dados Python, que é enviado aos microsserviços de tradução.

A camada controle de acesso é baseada em papéis para administradores com diferentes níveis de acesso aos recursos da FWunify. Esta camada foi implementada utilizando o módulo de controle de acesso do Flask, o Flask-RBAC[2] (*Role-based Access Control*), que permite a definição de diferentes funções dentro da aplicação. Para fins de testes, foram predefinidos três níveis de acesso. Administradores do primeiro nível conseguem realizar somente a tradução e aplicação de políticas de *traffic shaping*. Os administradores do nível intermediário podem traduzir e aplicar políticas de *traffic*

[1]<https://https://flask.palletsprojects.com/en/1.1.x//>
[2]<https://flask-rbac.readthedocs.io/en/latest/>

shaping e NAT 1to1. Por fim, os administradores do terceiro nível podem realizar qualquer operação suportada pela solução, como aplicar políticas de roteamento estático, NAT Nto1 e ACLs.

Uma vez liberadas pela camada de controle de acesso, as políticas de segurança seguem para a camada de resolução de conflitos onde são realizadas verificações de duplicidade e sobreposição de políticas. Para identificar violações, a camada de resolução de conflitos foi implementada pelo módulo *PyFwConflict*, que utiliza o histórico de políticas da FWunify. Se as novas políticas estão duplicando ou sobrepondo políticas existentes, o processo de tradução é abortado e uma notificação é enviada ao administrador do sistema. A verificação de sobreposição de políticas tem por objetivo evitar que duas políticas diferentes tratem um mesmo tipo de tráfego. Por exemplo, adicionar uma regra que libere o tráfego FTP para o IP 10.0.0.5, sendo que existe outra regra que libera o tráfego para o bloco de IPs 10.0.0.0/24.

Os quatro tradutores, para Cisco ASA, IPTables, OpenFlow (Open vSwitch) e Palo Alto, representam a camada microsserviços de tradução. A comunicação das camadas superiores com os microsserviços é realizada através de chamadas remotas de procedimento, ou RPC (*Remote Procedure Call*). Para realizar as chamadas remotas, a API da camada interface norte deve conhecer os microsserviços de tradução disponíveis e habilitados no sistema, bem como as funções disponíveis em cada microsserviço (ver Apêndice B). As chamadas remotas são definidas no padrão URI_AMPQ[3], incluindo detalhes de autenticação, IP e porta do servidor que disponibiliza o microsserviço. Para o processo de tradução, cada microsserviço deve receber um dicionário de dados com as informações extraídas da intenção. Por exemplo, para uma política do tipo ACL são obrigatórias informações como origem e destino dos pacotes, ação a ser tomada, e a ordem de prioridade da regra no respectivo firewall.

O tradutor então recebe o dicionário de dados e realiza a tradução para a sintaxe de comandos específicos do firewall. O processo de tradução pode incluir, inclusive, a conversão de unidades, como a conversão da largura de banda de Mbps para Kbps. O processo de tradução é realizado com o suporte da linguagem de modelagem de *templates* Jinja2[4]. Esta linguagem permite criar *templates* contendo partes fixas e dinâmicas das sintaxes dos comandos para os diferentes firewalls. As partes dinâmicas (*e.g.*, endereços IP, largura de banda) são preenchidas automaticamente pela Jinja2 a partir do dicionário de dados recebido da camada subjacente, gerando os comandos a serem aplicados nos

[3] <https://www.rabbitmq.com/uri-spec.html>
[4] <https://jinja.palletsprojects.com/en/2.11.x/>

dispositivos de firewall.

Por fim, os microsserviços de tradução utilizam a interface sul, como os conectores SSH, para realizar a aplicação das regras nas respectivas soluções de firewall. Os tradutores realizam a chamada remota de procedimento para o conector definido na implementação, utilizando o padrão URI_AMPQ, assim como utilizado na camada anterior. Os conectores recebem como parâmetro o endereço IP de gerência para acesso ao dispositivo, credenciais de acesso (*i.e.*, usuário e senha) e a sequência dos comandos a serem aplicados ao firewall. Na implementação atual, há dois conectores SSH, um específico para equipamentos Cisco da série ASA 5020 e outro para os sistemas Linux que implementam os firewalls IPTables e OpenFlow. Os conectores SSH foram implementados através da biblioteca Netmiko[5], que utiliza parâmetros como endereço IP, porta e usuário para estabelecer as conexões SSH. Os conectores retornam para o microsserviço de tradução as informações de estado (*e.g.*, sucesso ou falha) da aplicação da política na solução de firewalls.

[5]<https://github.com/ktbyers/netmiko>

6 AVALIAÇÃO

Neste capítulo, apresentamos o ambientes de testes e os resultados da avaliação da implementação da arquitetura FWunify e da linguagem FWlang. O ambiente de testes é composto por uma rede híbrida contendo três tipos diferentes de firewall, como detalhado na Seção 6.1. Na avaliação da FWunify e FWlang, analisamos aspectos como: (a) redução de complexidade na representação de políticas reais de segurança (Seção 6.2); (b) intuitividade e complexidade na utilização da FWlang em relação à sintaxes específicas dos firewall (Seção 6.3); (c) corretude do processo de tradução das intenções em regras específicas dos firewalls (Seção 6.4); (d) eficácia das regras geradas e aplicadas automaticamente nos firewalls (Seção 6.5); e (e) aplicação de múltiplas intenções de segurança com restrições de ordem e prioridade entre as regras (Seção 6.6).

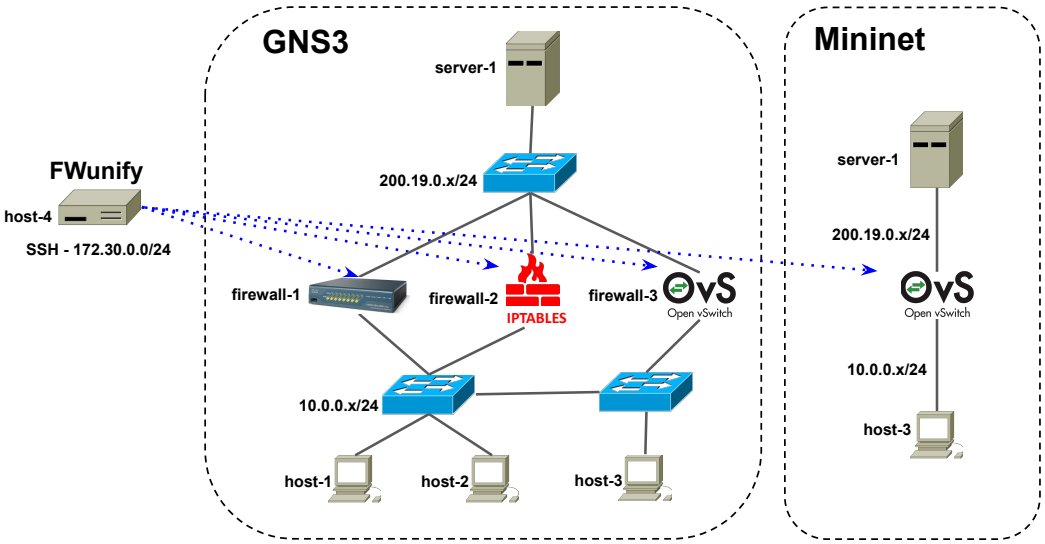
6.1 Ambientes de testes

Os testes realizados com a solução FWunify foram realizados em dois ambientes, conforme ilustrado na Figura 8. O primeiro, uma rede híbrida definida no GNS3[1], é composto por: (a) três firewalls – Cisco ASA-5520 (firewall-1), IPTables (firewall-2) e Open vSwitch (firewall-3) – posicionados entre as redes interna (10.0.0.0/24) e externa (200.19.0.0/24) do cenário; (b) um servidor Ubuntu Server 18.04 (*server-1*), conectado logicamente as interfaces externas dos firewalls; (c) três *hosts* executando Lubuntu 16.04 (*host-1, host-2* e *host-3*), que utilizam, respectivamente, os firewalls 1, 2 e 3 como *gateway* principal; e (d) um *host* Ubuntu 18.04 (*host-4*) executando a FWunify e conectado aos firewalls por uma interface específica para gerenciamento.

O GNS3 possui uma limitação para testes de desempenho, mais especificamente com relação à largura de banda[2]. As conexões disponibilizadas pelo GNS3 são limitadas a larguras de banda baixas (de 1 a 4 Mbps dependendo do sistema hospedeiro). Além disso, segundo resultados empíricos externos e próprios, a vazão oscila de uma forma imprevisível, inviabilizando testes de desempenho mais confiáveis. Consequentemente, para realizar uma demonstração de uma intenção de *traffic shaping* na prática, foi criado um segundo ambiente de testes, utilizando o Mininet[3]. O ambiente é composto pelo

[1] <https://www.gns3.com/>
[2] <https://tinyurl.com/gns3-limitation>
[3] <http://mininet.org/>

Figura 8 – Ambientes de testes



host-4, host-3, firewall-3 e *server-1*, conforme ilustrado na Figura 8.

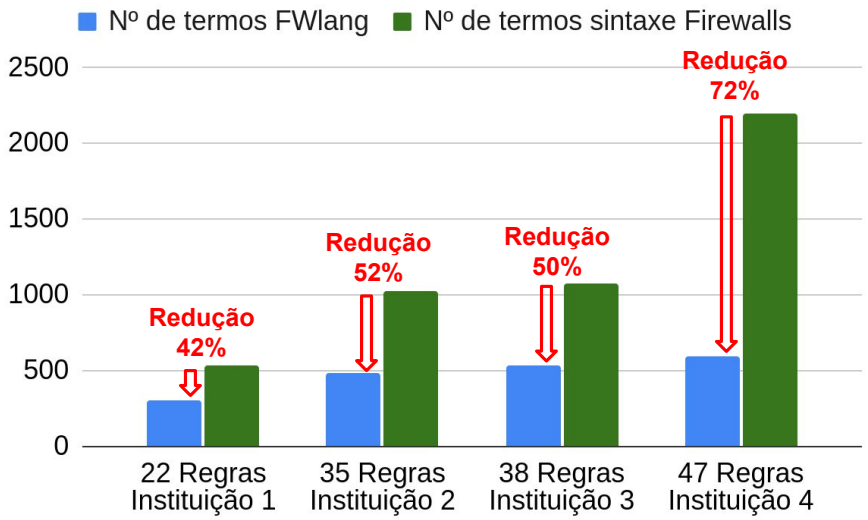
6.2 Redução de Complexidade

Considerando o contexto da diversidade de sintaxe, a inerente complexidade e propensão a erros humanos, como apontado por relatórios técnicos e estudos especializados (Gartner, 2019; VORONKOV; MARTUCCI; LINDSKOG, 2019; IBM X-Force, 2020; Oracle, 2020), linguagem genéricas baseadas em intenções de nível mais abstrato (*i.e.*, sem precisar conhecer detalhes técnicos da rede e das sintaxes específicas de diferentes firewalls), como a FWlang, representam uma forma de simplificar a representação de políticas de segurança e automatizar o processo de tradução e aplicação das regras nos firewalls. Para demonstrar o potencial de simplificação e redução da propensão a erros, foram analisados conjuntos de regras em ambiente de produção de quatro instituições, sendo compostos por: (a) 22 políticas ACL de um firewall interno que controla o tráfego entre filiais (Shopos); (b) 35 políticas ACL de um firewall protegendo um servidor Web em um data center compartilhado (IPTables); (c) 38 políticas ACL de um firewall de borda (Cisco); e (d) 47 políticas de um segundo firewall de borda, sendo 8 ACLs e 39 regras de NAT 1to1 (MikroTik).

Para efeitos de comparação, consideramos o número de termos fixos (da sintaxe) necessários para a descrição e aplicação das políticas em três firewalls diferentes (Cisco,

IPTalbles e OpenFlow), utilizando a FWlang e as sintaxes específicas de cada solução. No gráfico da Figura 9 podemos observar uma redução no número de termos utilizados para aplicar as políticas através de intenções da FWlang. A comparação leva em consideração os comandos necessários para aplicação manual das regras nos firewalls tradicionais.

Figura 9 – Número de termos em diferentes sintaxes



A redução no número de termos para os quatro conjuntos de regras foi de 42%, 52%, 50% e 72%. Essa redução mais significativa no quarto conjunto de regras é resultado do número elevado de termos utilizados para descrição das políticas de NAT. É importante ressaltar também que o número de termos cresce exponencialmente à medida que novas soluções de firewall são adicionadas ao ambiente.

Os números indicam que a FWlang simplifica a representação de políticas de firewalls. Quanto mais diversificado for o conjunto de soluções de firewall a serem gerenciadas, maior será a simplificação da gestão uma vez que o administrador dos sistemas precisa compreender e representar as intenções de segurança uma única vez, utilizando a sintaxe da FWlang. Na prática, mesmo a instituição possuindo apenas firewalls Cisco ASA 5500, com versões de software distintas (como a 8.2 e a 8.4), onde as sintaxes dos comandos variam significativamente (*e.g.*, podem ter o dobro de número de termos), os administradores dos sistema precisam representar uma única vez as intenções de segurança com a FWlang e a FWunify.

6.3 Intuitividade da FWlang

Com o objetivo de coletar as impressões de administradores de sistemas sobre a linguagem FWlang, criamos um questionário online[4]. No questionário, que foi enviado para profissionais com experiência em administração de sistemas e/ou firewalls, há questões que objetivam identificar a intuitividade[5] e complexidade[6] da linguagem quando comparada com exemplos utilizando sintaxes específicas de firewalls atuais.

Seguindo o exemplo de pesquisas recentes que avaliam a utilização de linguagens baseadas em intenções pelos usuários (SCHEID et al., 2020), no questionário online utilizamos três políticas de segurança (Filtro de URL, *Traffic Shaping* e ACL) para guiar a avaliação dos usuários. Todas as políticas são apresentadas em FWlang e na sintaxe específica dos firewalls. Para cada representação das políticas, o participante indica nível de intuitividade (em uma escada de 1 a 5, onde 1 significa "Pouco intuitiva" e 5 significa "Muito intuitiva") e o nível de complexidade (também em uma escada de 1 a 5, onde 1 significa "Pouco complexa" e 5 significa "Muito complexa") da representação. Antes da aplicação, o questionário passou pela revisão de três especialistas em gerenciamento de redes e/ou firewalls (LINAKER et al., 2015). A análise observou pontos como o nível técnico e o fluxo de apresentação das questões, a corretude e inteligibilidade do texto técnico das questões, e o alinhamento da instrumentação utilizada com os objetivos da pesquisa.

Em um intervalo de vinte dias, tivemos a contribuição de 22 administradores de sistemas, a maioria com curso superior (27%), especialização (36%) ou mestrado (18%) em computação. Os demais (19%), possuem curso técnico ou doutorado em computação. Pouco mais da metade dos respondentes (59%) relataram alguma experiência no gerenciamento prático de firewalls como IPTables, pfSense, Cisco, Sophos, Mikrotik e IPFilter. Os demais (41%), atuam em administração de sistemas, mas não diretamente com equipamentos de firewall. Este segundo grupo de participantes é interessante pelo fato de não possuir "vícios" que podem levar a uma avaliação potencialmente tendenciosa das sintaxes de representação de políticas de segurança. Por exemplo, um administrador de firewalls IPTables ou Cisco pode, dada a sua experiência prévia e memória "mecânica" das sintaxes, tender a uma avaliação "viciada", isto é, que considera as sintaxes dos

[4]<https://forms.gle/iSQt27j26Xcnp2hT6>

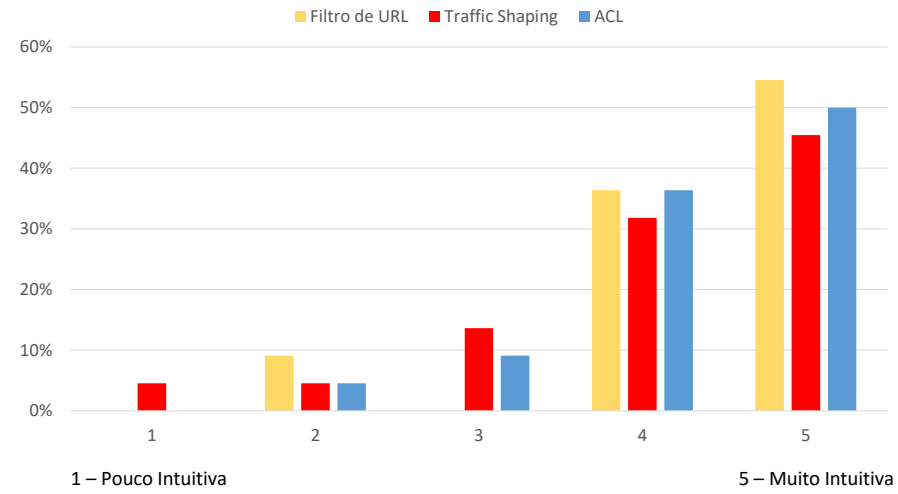
[5]Intuitividade considerando aspectos como facilidade de interpretação de todos os termos, nível de conhecimento prévio exigido para interpretação e facilidade para memorização.

[6]Complexidade considerando aspectos como quantidade de termos utilizados, verbosidade e legibilidade.

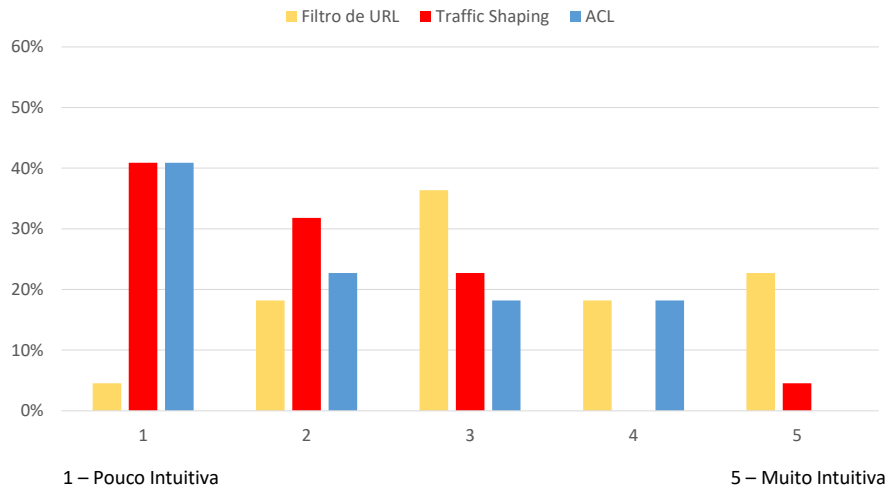
firewalls conhecidos automaticamente simples, intuitiva e pouco complexa.

Os gráficos da Figura 10 resumem o nível de intuitividade indicado pelos participantes da pesquisa. Para cada opção da escada de 1 a 5, as barras dos gráficos indicam a porcentagem de respondentes que atribuiu aquele valor. Como pode ser observado na Figura 10(a), a maioria absoluta dos participantes considera a sintaxe da FWlang muito intuitiva (níveis 4 e 5) para os três cenários de políticas de segurança ACL, *traffic shaping* e filtro de URL. É importante observar que avaliação é similar, independente do cenário, isto é, número de sintaxes distintas envolvidas na representação. Por exemplo, para o cenário de *traffic shaping* é necessário configurar dois firewalls distintos, enquanto que no cenário de ACL são configurados cinco firewalls distintos.

Figura 10 – Análise da intuitividade das sintaxes



(a) FWlang



(b) Outras sintaxes

Ao observamos o gráfico da Figura 10(b), podemos constatar que a maioria dos

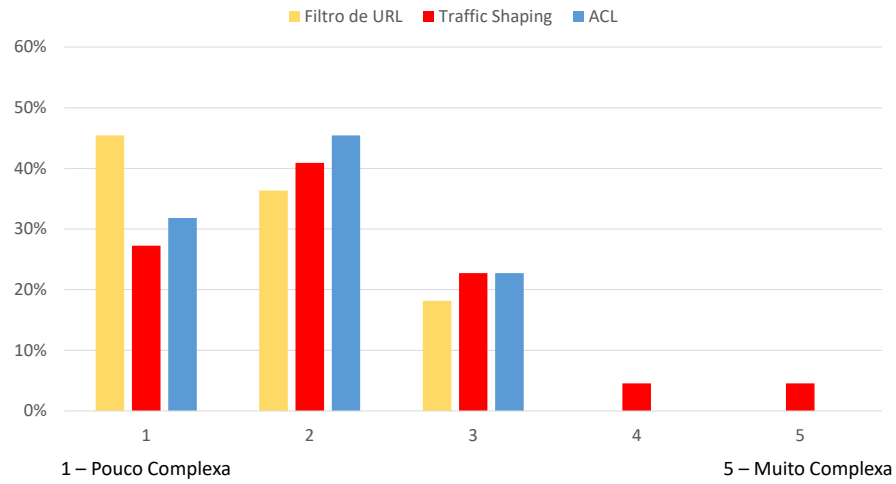
participantes classifica as representações em sintaxes específicas dos firewalls pouco intuitiva (níveis 1, 2 e 3) quando comparadas com a FWlang. Vejam que isto ocorre, inclusive, para a sintaxe do cenário de filtro de URL, que é o mais simples de todos, onde a sintaxe da FWlang é comparada com a sintaxe de um firewall Palo Alto. Apesar de a maioria (57%) dos participantes possuir experiência com firewalls, mais de 60% dos administradores de sistemas entrevistados julga a sintaxe específica do firewall pouco intuitiva (níveis 3, 2 ou 1) quando comparada com a FWlang (gráfico da Figura 10(a)). Esta diferença acentua-se com o aumento no número e variabilidade de termos das sintaxes, como pode ser visto para as regras de *traffic shaping* e ACL, que são representadas em múltiplas sintaxes específicas de firewalls. É importante observar que, mesmo no caso das políticas de ACL, que podem ser consideradas o tipo mais comum e conhecido de políticas de segurança[7], a intuitividade reduz significativamente quando expressadas em diferentes sintaxes de firewalls. Mesmo considerando que 57% dos participantes possuem experiência com gerenciamento de firewalls, mais de 75% dos entrevistados considera as políticas ACL apresentadas no questionário, na sintaxe específica dos equipamentos, pouco intuitiva.

Com relação à complexidade das representações nas diferentes sintaxes, podemos observar um resultado similar, como apresentado nos gráficos da Figura 11. Similar à análise de intuitividade, para a FWlang (gráfico da Figura 11(a)), os participantes atribuíram uma complexidade baixa para as representações das intenções de segurança. Para a sintaxe dos demais firewalls, a complexidade foi aumentando gradualmente de acordo com os cenários do questionário. Como pode ser observado, o cenário menos complexo foi o da política de filtro de URL, representada apenas na sintaxe de firewalls da família Palo Alto. Entretanto, mesmo neste cenário mais simples (uma única política representada em uma única sintaxe), a maioria dos participantes identificaram uma complexidade significativamente maior para a sintaxe Palo Alto quando comparada com a FWlang. Em síntese, a maioria dos respondentes identificou uma complexidade menor para a sintaxe FWlang (entre 1 e 2) e, ao mesmo tempo, uma complexidade maior (entre 3 e 4) para a sintaxe Palo Alto.

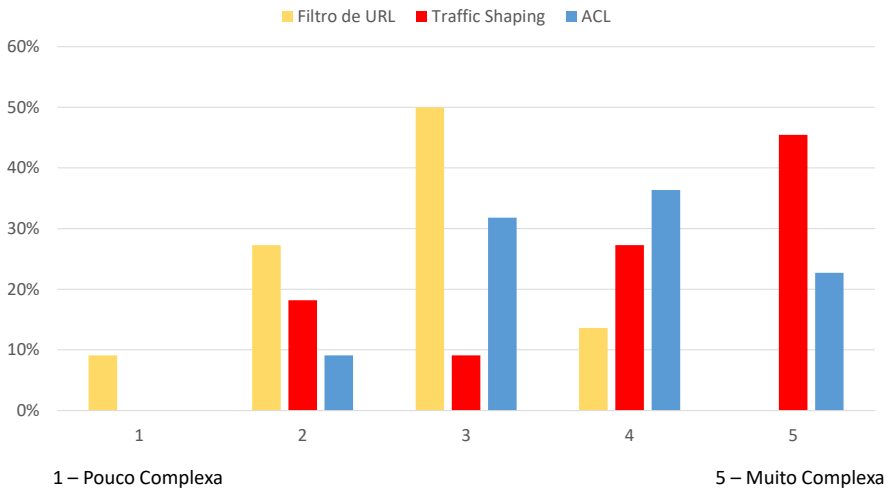
Com relação à complexidade da política de ACL, representada em cinco sintaxes diferentes, a explicação do comportamento do gráfico pode estar atrelada ao nível de

[7]Há soluções de segurança, linguagens e firewalls que implementam essencialmente políticas do tipo ACL, como pode ser observado na literatura (TIAN et al., 2019; POZO; VARELA-VACA; GASCA, 2009; ZHANG et al., 2007; BARTAL et al., 2004) e nos exemplos reais de políticas de segurança apresentados na Seção 6.2.

Figura 11 – Análise da complexidade das sintaxes



(a) FWlang



(b) Outras sintaxes

conhecimento dos respondentes. Como 57% dos respondentes possuem experiência em configuração de firewalls, e políticas do tipo ACL são as mais comuns e frequentemente utilizadas[7], há uma tendência natural em atribuir uma ligeiramente menor complexidade, aumentando o número de incidências nos níveis 3 e 4 de complexidade. Diferentemente, políticas de *traffic shaping* são menos conhecidas e aplicadas na prática, um dos motivos pelos quais pode ser observada uma acentuada porcentagem dos participantes classificando a política como muito complexa, isto é, atribuindo o nível máximo (5) de complexidade.

6.4 Corretude do processo de tradução

A avaliação da corretude do processo tradução das intenções pode ser realizada através da especificação, tradução automática e consequente verificação manual das regras (ou instruções de baixo nível) geradas, análogo ao realizado em trabalhos existentes na literatura (JACOBS et al., 2018; ZHANG et al., 2007; Al-Shaer; Hamed, 2003). Para avaliar a corretude da tradução utilizando a FWlang e a solução FWunify, foram definidas três tipos de políticas de segurança. A primeira, do tipo ACL, realiza o bloqueio do tráfego ICMP (*Internet Control Message Protocol*) entre as redes 10.0.0.0/24 e 200.19.0.0/24. A segunda, remove uma regra NAT 1to1 que realiza o redirecionamento do tráfego recebido na porta 80 do IP (externo) 200.19.0.50, para a porta 90 do IP (interno) 10.0.0.50. Por fim, uma política de *traffic shaping* é adicionada aos firewalls para limitar a 30Mbps o tráfego FTP entre a rede 10.0.0.0/24 e o IP 200.19.0.100.

A Figura 12 apresenta as três intenções em FWlang, o comando esperado e o comando gerado pela FWunify para firewalls Cisco ASA 5520. Os comandos esperados são provenientes da documentação oficial do fabricante[8]. Os resultados das traduções para os firewall IPTables e OpenFlow (Open vSwitch) estão disponíveis no Apêndice C, com exceção da regra de *traffic shaping* para IPTables, uma vez que este filtro de pacotes não suporta este tipo de política nativamente. Como pode ser observado na tradução das intenções (a), (b) e (c) da figura, os comandos resultantes do processo de tradução atendem rigorosamente a sintaxe da documentação do fabricante. Além disso, é importante ressaltar que todos os comandos gerados foram testados e validados manualmente nos respectivos firewalls.

Figura 12 – Tradução de intenções FWlang em comandos Cisco ASA 5520

Intenção em FWlang:
define intent ***acl****:*
name *text(**rule-acl-1**)*
from *range(**10.0.0.0/24**)*
to *range(**200.19.0.0/24**)*
order *before(**all-intents**)*
deny *traffic(**icmp**)*
add *firewall(**cisco-1**,**iptables-1**,**openflow-1**)*

Sintaxe do comando esperado:
*access-list "**interface**"_access_in line "**posição**" extended "**permitir/bloquear**" "**protocolo**" "**IP/rede de origem**" "**IP/rede de destino**"*

Comando gerado:
*access-list **inside**_access_in line **1** extended **deny icmp 10.0.0.0 255.255.255.0 200.19.0.0 255.255.255.0***

(a) *Tradução da intenção ACL*

[8] <https://tinyurl.com/cisco-asa-acl>, <https://tinyurl.com/cisco-asa-nat>, <https://tinyurl.com/cisco-asa-traffic>

Intenção em FWlang:
define intent ***nat_1to1****:*
*name text(**nat-200.19.0.50-10.0.0.50**)*
*from endpoint(**200.19.0.50**)*
*to endpoint(**10.0.0.50**)*
*for port(protocol:**tcp**;scr_port:**80**;dst_port:**90**)*
*del firewall(**cisco-1**,**iptables-1**,**openflow-1**)*

Sintaxe do comando esperado:
*object network "**IP válido**"*
*no nat static "**IP interno**" service "**protocolo**" "**porta origem**" "**porta destino**"*
*no object network "**IP válido**"*
*no object network "**IP interno**"*

Comando gerado:
*object network **200.19.0.50***
*no nat static **10.0.0.50** service **tcp 80 90***
*no object network **200.19.0.50***
*no object network **10.0.0.50***

(b) *Tradução da intenção NAT 1to1*

Intenção em FWlang:
define intent ***traffic_shaping****:*
*name text(**rule-ts-1**)*
*from range(**10.0.0.0/24**)*
*to endpoint(**200.19.0.100**)*
*order before(**all-intents**)*
*for traffic(**ftp**)*
*with throughput(**30Mbps**)*
*add firewall(**cisco-1**,**openflow-1**)*

Sintaxe do comando esperado:
*access-list global_mpc line **"posição"** extended permit **"protocolo" "IP/rede de origem" "IP/Rede de destino"***
*eq **"porta"***
*access-list global_mpc line **"posição"** extended permit **"protocolo" "IP/rede de destino" "IP/Rede de origem"***
*eq **"porta"***
*class-map **"nome class-map"***
match access-list global_mpc
policy-map global-policy
*class **"nome class-map"***
*police input **"largura de banda" "burst size"** conform-action transmit exceed-action drop*
*police output **"largura de banda" "burst size"** conform-action transmit exceed-action drop*

Comando gerado:
*access-list global_mpc line **1** extended permit **tcp 10.0.0.0 255.255.255.0 host 200.19.0.100** eq **21***
*access-list global_mpc line **2** extended permit **tcp host 200.19.0.100 10.0.0.0 255.255.255.0** eq **21***
*class-map global-class-**rule-ts-1***
match access-list global_mpc
policy-map global-policy
*class global-class-**rule-ts-1***
*police input **30000000 15000** conform-action transmit exceed-action drop*
*police output **30000000 15000** conform-action transmit exceed-action drop*

(c) *Tradução da intenção traffic shaping*

6.5 Eficácia das políticas aplicadas

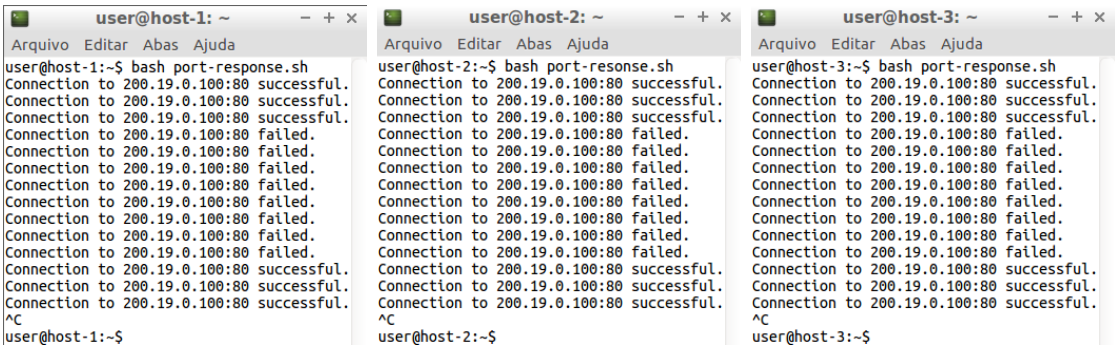
A avaliação da eficácia das instruções de baixo nível, geradas a partir de uma descrição de mais alto nível, como as intenções utilizadas em redes baseadas no paradigma de IBN, pode ser realizada através da tradução, aplicação automática e avaliação das políticas utilizando cenários e ferramentas externas, similar ao realizado em trabalhos existentes na literatura (RIFTADI; KUIPERS, 2019; SOULE et al., 2018; SUBRAMANYA; RIGGIO; RASHEED, 2016). Para realizar a avaliação da eficácia da tradução e aplicação das intenções de segurança utilizando a FWlang e a FWunify, foram

especificadas duas políticas, uma do tipo ACL e outra do tipo *traffic shaping*.

A ACL utilizada, que pode ser vista na Figura 6(a), interrompe o tráfego HTTP (porta 80/TCP) entre as máquinas *host-1*, *host-2* e *host-3* e o servidor *server-1* do ambiente de testes descrito na Seção 6.1. A intenção foi traduzida e enviada aos três firewalls (Cisco, IPTables e OpenFlow) utilizando a FWlang e FWunify. Após aplicar a ACL, para testar a conectividade na porta 80/TCP do *server-1*, foi executado o um *shell script* Bash (*port-response.sh* – disponível junto ao código fonte da solução FWunify) que utiliza o comando *netcat*[9] do Linux. O *script* é programado para realizar uma tentativa de conexão a cada 5 segundos.

As Figuras 13(a), 13(b) e 13(c) apresentam os resultados da execução do *script* em cada um dos hosts, antes da aplicação da política, durante o período em que a política estava ativa e após a remoção da regra nos respectivos firewalls. Como pode ser verificado, durante o período de vigência da política, nenhum dos hosts conseguiu realizar a conexão com o servidor. Os detalhes da inserção e remoção da intenção utilizando a FWunify estão disponíveis no Apêndice D.

Figura 13 – Intenção ACL: comportamento das conexões



(a) Execução do *script* no host-1 (b) Execução do *script* no host-2 (c) Execução do *script* no host-3

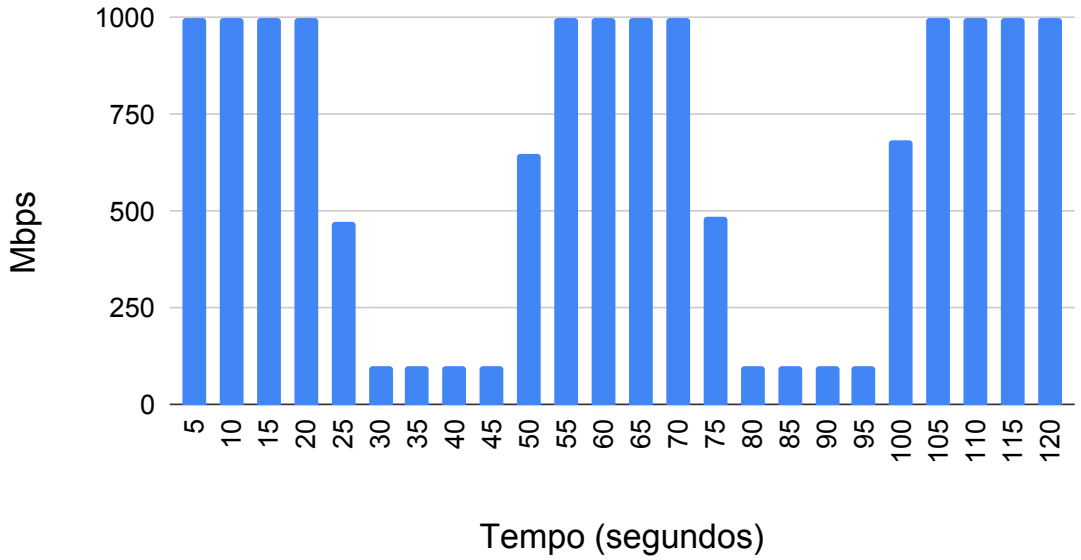
No segundo teste foi especificada e aplicada a intenção de *traffic shaping* apresentada na Figura 6(b). Essa intenção limita o tráfego da porta 5555/UDP a 100Mbps (`throughput(100Mbps)`) entre a rede do *host-3* (10.0.0.0/24) e o servidor *server-1*, cujo link é de 1Gbps. A conexão entre o host e o servidor é estabelecida através do Open vSwitch (OpenFlow versão 1.3).

A medição da vazão entre o host e o servidor foi realizada com o auxílio da ferramenta *iperf*[10]. O gráfico da Figura 14 apresenta o comportamento da vazão da conexão antes, durante e depois de duas aplicações e remoções consecutivas da intenção

[9] <https://docs.oracle.com/cd/E86824_01/html/E54763/netcat-1.html>
[10] <https://iperf.fr/>

de *traffic shaping*. Como pode ser observado, a vazão reduz para 100Mbps quando a intenção é aplicada ao firewall e retorna a 1Gbps após a remoção da intenção. Os registros (*logs*) e detalhes de execução do teste estão disponíveis no Apêndice E.

Figura 14 – Intenção de *Traffic Shaping*: comportamento do tráfego



6.6 Aplicação de múltiplas intenções nos firewalls

Para demonstrar o correto funcionamento da tradução e da aplicação de políticas de segurança nos firewalls, levando em consideração requisitos importantes como a ordem das regras resultantes, foram definidos dois conjuntos de sequências de intenções para serem adicionadas nos firewalls. O primeiro conjunto, apresentado a seguir, é composto por seis intenções do tipo ACL inspiradas nas regras em uso no firewall de borda (Cisco) de uma das instituições analisadas.

1. Intenção “drop-all-all”: bloqueia o tráfego de todas as origens e destinos.

2. Intenção “permit-net-all-http”: libera o tráfego da rede interna (10.0.0.0/24) para qualquer destino pela porta 80/TCP (HTTP). Esta intenção deve ter prioridade sobre todas as existentes.

3. Intenção “permit-net-all-https”: libera o tráfego da rede interna (10.0.0.0/24) para

qualquer destino pela porta 443/TCP (HTTPS). Esta intenção deve ter prioridade sobre todas as existentes.

4. Intenção “permit-h10-h20-mysql”: libera conexões à porta 3306/TCP (MySQL), do IP 200.19.0.10, para o IP 10.0.0.10. Esta intenção deve prescindir a intenção “drop-all-all”.

5. Intenção “drop-net-h20-mysql”: bloqueia as conexões da rede interna (10.0.0.0/24) ao servidor h20 (200.19.0.10) e porta 3306/TCP. Esta intenção deve ser adicionada após a intenção “permit-h10-h20-mysql”.

6. Intenção “drop-incident-h21”: bloqueia todo o tráfego com destino ao servidor h21 (200.19.0.20), cuja origem esteja envolvida em algum incidente de segurança. Esta intenção deve ter prioridade máxima em relação às demais.

As seis intenções de segurança devem ser aplicadas aos três firewalls do ambiente de testes, Cisco, IPTables e OpenFlow. A tradução das intenções e inserção das regras resultantes nos firewalls deve seguir a ordem de prioridade estabelecida nas ACLs. Para verificar a correta tradução e aplicação das intenções das ACLs, as regras foram geradas também manualmente para cada um dos firewalls.

A Figura 15 resume as ACLs nas regras definidas manualmente para a sintaxe específica de cada firewall. Essas regras foram inseridas manualmente nos firewalls através das respectivas interfaces de gerenciamento via linha de comando. A Figura 16 apresenta as políticas de segurança 4 e 5 no formato de intenções FWlang, que são utilizadas para gerar e aplicar as regras automaticamente nos firewalls através da solução FWunify. O conjunto completo das seis regras representadas em FWlang pode ser visualizado no Apêndice F. As intenções da figura representam as regras “permit-h10-h20-mysql” e “drop-net-h20-mysql”, onde podemos observar a utilização do *order* para definição da prioridade de processamento das regras no firewall. A intenção “permit-h10-h20-mysql” deverá ser adicionada antes da regra “drop-all-all” (*order before(drop-all-all)*), enquanto que a intenção “drop-net-h20-mysql” deve suceder a regra gerada pela intenção anterior (*order after(permit-h10-h20-mysql)*).

Os resultados da inserção manual e automática (via FWunify) das regras no firewall Cisco podem ser vistos na Figura 17. Os resultados para os firewalls IPTables e Open vSwitch estão disponíveis no Apêndice G. Nos três casos, as inserções manuais e via protótipo geraram resultados idênticos, conforme o esperado, isto é, apresentam exatamente o mesmo comportamento funcional.

Figura 15 – Regras geradas e inseridas manualmente

Cisco:
- access-list inside_access_in extended deny ip any any
- access-list inside_access_in line 1 extended permit tcp 10.0.0.0 255.255.255.0 any eq 80
- access-list inside_access_in line 1 extended permit tcp 10.0.0.0 255.255.255.0 any eq 443
- access-list inside_access_in line 3 extended permit tcp host 10.0.0.10 host 200.19.0.10 eq 3306
- access-list inside_access_in line 4 extended deny tcp 10.0.0.0 255.255.255.0 host 200.19.0.10 eq 3306
- access-list inside_access_in line 1 extended deny ip any host 200.19.0.20

IPTables:
- iptables -A FORWARD -s 0.0.0.0/0.0.0.0 -d 0.0.0.0/0.0.0.0 -p all -j DROP
- iptables -I FORWARD 1 -s 10.0.0.0/255.255.255.0 -d 0.0.0.0/0.0.0.0 -p tcp --dport 80 -j ACCEPT
- iptables -I FORWARD 1 -s 10.0.0.0/255.255.255.0 -d 0.0.0.0/0.0.0.0 -p tcp --dport 443 -j ACCEPT
- iptables -I FORWARD 3 -s 10.0.0.10 -d 200.19.0.100 -p tcp --dport 3306 -j ACCEPT
- iptables -I FORWARD 4 -s 10.0.0.0/255.255.255.0 -d 200.19.0.100 -p tcp --dport 3306 -j DROP
- iptables -I FORWARD 1 -s 0.0.0.0/0.0.0.0 -d 200.19.0.20 -p all -j DROP

OpenFlow:
- ovs-ofctl add-flow br0 dl_type=0x800,priority=15000,nw_src=0.0.0.0/0.0.0.0,nw_dst=0.0.0.0/0.0.0.0, action=drop
- ovs-ofctl add-flow br0 dl_type=0x800,priority=32000,nw_src=10.0.0.0/255.255.255.0,nw_dst=0.0.0.0/0.0.0.0, nw_proto=6,tcp_dst=80,action=normal
- ovs-ofctl add-flow br0 dl_type=0x800,priority=32100,nw_src=10.0.0.0/255.255.255.0,nw_dst=0.0.0.0/0.0.0.0, nw_proto=6,tcp_dst=443,action=normal
- ovs-ofctl add-flow br0 dl_type=0x800,priority=31900,nw_src=10.0.0.10,nw_dst=200.19.0.10,nw_proto=6, tcp_dst=3306,action=normal
- ovs-ofctl add-flow br0 dl_type=0x800,priority=31800,nw_src=10.0.0.0/255.255.255.0,nw_dst=200.19.0.10, nw_proto=6,tcp_dst=3306,action=drop
- ovs-ofctl add-flow br0 dl_type=0x800,priority=32200,nw_src=0.0.0.0/0.0.0.0,nw_dst=200.19.0.20,action=drop

Figura 16 – Políticas de segurança 4 e 5 em FWlang

```
define intent acl:
name        text(permit-h10-h20-mysql)
from        endpoint(10.0.0.10)
to          endpoint(200.19.0.10)
allow       traffic(tcp/3306)
order       before(drop-all-all)
add         firewall(cisco-1,iptables-1,openflow-1)
```

(a) *Intenção 4*

```
define intent acl:
name        text(deny-net-h20-mysql)
from        range(10.0.0.0/24)
to          endpoint(200.19.0.10)
deny        traffic(tcp/3306)
order       after(permit-h10-h20-mysql)
add         firewall(cisco-1,iptables-1,openflow-1)
```

(b) *Intenção 5*

Figura 17 – Resultados das inserções no firewall Cisco

```
firewall-1-cisco#
firewall-1-cisco# show access-list
access-list cached ACL log flows: total 0, denied 0 (deny-flow-max 4096)
            alert-interval 300
access-list inside_access_in; 6 elements; name hash: 0x433a1af1
access-list inside_access_in line 1 extended deny ip any host 200.19.0.20 (hitcnt=0) 0xfdb26f1a
access-list inside_access_in line 2 extended permit tcp 10.0.0.0 255.255.255.0 any eq https (hitcnt=0) 0xec5c9adf
access-list inside_access_in line 3 extended permit tcp 10.0.0.0 255.255.255.0 any eq www (hitcnt=0) 0xa26ab2db
access-list inside_access_in line 4 extended permit tcp host 10.0.0.10 host 200.19.0.10 eq 3306 (hitcnt=0) 0xd683cdb3
access-list inside_access_in line 5 extended deny tcp 10.0.0.0 255.255.255.0 host 200.19.0.10 eq 3306 (hitcnt=0) 0x16c38c22
access-list inside_access_in line 6 extended deny ip any any (hitcnt=0) 0xbe9efe96
firewall-1-cisco#
```

(a) *Inserção manual*

```
firewall-1-cisco#
firewall-1-cisco# show access-list
access-list cached ACL log flows: total 0, denied 0 (deny-flow-max 4096)
            alert-interval 300
access-list inside_access_in; 6 elements; name hash: 0x433a1af1
access-list inside_access_in line 1 extended deny ip any host 200.19.0.20 (hitcnt=0) 0xfdb26f1a
access-list inside_access_in line 2 extended permit tcp 10.0.0.0 255.255.255.0 any eq https (hitcnt=0) 0xec5c9adf
access-list inside_access_in line 3 extended permit tcp 10.0.0.0 255.255.255.0 any eq www (hitcnt=0) 0xa26ab2db
access-list inside_access_in line 4 extended permit tcp host 10.0.0.10 host 200.19.0.10 eq 3306 (hitcnt=0) 0xd683cdb3
access-list inside_access_in line 5 extended deny tcp 10.0.0.0 255.255.255.0 host 200.19.0.10 eq 3306 (hitcnt=0) 0x16c38c22
access-list inside_access_in line 6 extended deny ip any any (hitcnt=0) 0xbe9efe96
firewall-1-cisco#
```

(b) *Inserção via FWunify*

O segundo conjunto contempla 38 regras ativas no firewall de borda (Cisco) atualmente utilizado pela mesma instituição do conjunto de regras anterior. As políticas foram descritas no formato de intenções FWlang e adicionadas ao firewall-1 do cenário de testes. Os detalhes das intenções FWlang e regras geradas manualmente (na CLI do firewall) e automaticamente (via FWunify) são apresentados no Apêndice H. Mais uma vez, a solução FWunify gerou as regras na ordem e formato esperados, ou seja, resultado idêntico à definição e aplicação manual das regras, em uso no firewall Cisco, em ambiente de produção, da respectiva instituição.

7 CONSIDERAÇÕES FINAIS

Neste capítulo, iniciamos com um resumo das principais contribuições do trabalho. Na sequência, apresentamos alguns exemplos de trabalhos futuros e a lista de publicações diretamente relacionadas ao trabalho desta dissertação.

7.1 Contribuições

As principais contribuições deste trabalho são: (a) a definição da arquitetura de software FWunify; (b) a especificação da linguagem de representação de intenções de segurança FWlang; e (c) a implementação e disponibilização de uma versão funcional da arquitetura FWunify. A FWunify é uma arquitetura de software organizada em camadas e módulos que visa orientar o desenvolvimento de soluções de gerência de firewalls para redes híbridas. Para isso, a FWunify possui uma estrutura com sete camadas independentes, que são interconectadas e utilizadas nos processos de definição, tradução e aplicação das políticas de segurança de firewalls modernos.

A arquitetura requer uma linguagem genérica para representar de maneira unificada as regras de segurança aplicáveis em firewalls diferentes em termos de paradigma, fabricante, linha ou modelo. Para atender essa demanda, propomos a FWlang, uma linguagem baseada no conceito de intenções, que permite descrever regras de segurança de uma forma mais simples e intuitiva que o conceito de políticas.

A FWunify foi avaliada através de um processo empírico experimental baseado em *datasets* reais utilizados na solução implementada e disponibilizada publicamente em <https://github.com/mmfiorenza/fwunify>. Os resultados demonstram o funcionamento correto do processo de tradução de políticas de segurança, a efetividade das políticas traduzidas, e a corretude na geração e aplicação de regras em firewalls reais levando em consideração aspectos importantes como ordem e prioridade das regras. A FWlang, por sua vez, foi avaliada em critérios objetivos e subjetivos. Uma avaliação usando *datasets* reais demonstrou que a FWlang pode reduzir em até 72% no número de termos utilizados para especificar um conjunto de políticas de segurança para a configuração de diferentes tipos de firewalls. A avaliação subjetiva, utilizando um questionário online, identificou que a maioria dos participantes classificou a FWlang como mais intuitiva e menos complexa quando comparada com sintaxes específicas de firewalls.

7.2 Limitações e Dificuldades

A seguir, resumimos algumas das limitações e desafios do trabalho.

1. Questões relacionadas ao desempenho e a segurança da arquitetura de software FWunify não foram consideradas na implementação e nem na avaliação da solução. Avaliações de segurança e desempenho são um exemplo de trabalho futuro, isto é, continuidade no desenvolvimento da solução.

2. A implementação atual da FWunify carece de validadores de arquivos de configuração. Este tipo de arquivo é crítico para o correto funcionamento da solução.

3. A resistência (compreensível) dos administradores de firewalls em disponibilizar *datasets* de regras demandou tempo e persistência na etapa de avaliação da FWlang. Como as regras dos firewalls fazem parte da política interna de segurança das organizações, não é comum os administradores de firewalls divulgarem esses *datasets*, mesmo que seja para pesquisa científica e inovação. Apesar disso, com muito esforço, conseguimos acesso a quatro *datasets* corporativos, o que contribuiu para a análise apresentada na Seção 6.2.

4. Os fabricantes de soluções de firewall consolidadas, como Cisco, Check Point e Palo Alto, não disponibilizam versões emuladas online dos firewalls. Isto, naturalmente, dificulta o trabalho de validação da solução em termos de aplicação prática automática das regras de firewall geradas. Felizmente, tivemos acesso a um emulador off-line e um firewall físico da Cisco.

7.3 Trabalhos Futuros

Como trabalhos futuros, destacamos seis direções, como segue.

1. Desenvolver de uma interface amigável a usuários com pouca experiência na (ou pouco tempo para) gerência de firewalls (VORONKOV; MARTUCCI; LINDSKOG, 2019). Nesse contexto, podem ser utilizadas desde interfaces gráficas, até recursos como *chatbots* que permitam descrever políticas em linguagem natural (VORONKOV; MARTUCCI, 2020).

2. Incluir outros mecanismos e técnicas de verificação na camada de resolução de conflitos do FWunify. Exemplos vão desde a composição da regra (Liu; Gouda, 2008; Maldonado-Lopez; Calle; Donoso, 2015), passam pela análise das regras já instaladas nos firewalls (Cordova; Marcovich; Santivanez, 2018; AL-SHAER et al., 2006; SAâDAOUI; Ben Youssef Ben Souayeh; BOUHOULA, 2017), até a análise do comportamento da rede (HU et al., 2014; HU et al., 2019) como forma de evitar erros de configuração que causem danos a rede.

3. Projetar e implementar uma instância camada transversal de *Monitoramento e Inteligência*, isto é, incorporar algoritmos que permitam a retro-alimentação automática da solução, por exemplo. Essa camada pode ser considerada um passo a mais na direção de soluções de gerenciamento autônomo de firewalls.

4. Evoluir o FWunify para uma solução de IBF (*Intent-Based Firewalling*), incorporando conceitos presentes em IBNs (CLEMM et al., 2021; GAO; CONTRERAS; RANDRIAMASY, 2020), como (a) monitoramento constante do tráfego, através de telemetria, fornecendo à solução e ao operador o estado atual da rede como forma de suportar decisões; e, (b) controle automatizado da rede, utilizando aprendizagem de máquina e inteligência artificial como forma de mecanizar atividades de gestão, e garantir que a rede de fato atenda as intenções do operador.

5. Implementar um validador de arquivos de configuração utilizados na solução, facilitando o trabalho e assegurando o correto funcionamento da solução aos operadores do sistema. Soluções como o ConfigValidator (BASET et al., 2017) são exemplos de ferramentas que podem ser estendidas para suportar a validação dos arquivos de configuração da FWunify.

6. Expandir a avaliação do número de termos utilizados na descrição de políticas para incluir conjuntos de regras de outras instituições. Obter *datasets* representativos foi uma dificuldade encontrada durante a execução do trabalho, pois as organizações evitam liberar seus conjuntos de regras por questões de segurança.

7. Realizar uma avaliação abrangente da usabilidade da linguagem FWlang, levando em consideração fatores como curva de aprendizagem, retenção ao longo do tempo, taxa de erros por usuário e satisfação subjetiva (VORONKOV et al., 2017).

8. Extensão da linguagem FWlang para incorporar outros tipos de políticas de segurança específicas a linhas de equipamentos, como *DoS Protection* da Palo Alto.

7.4 Publicações

O trabalho apresentado resultou em três artigos científicos, elencados a seguir.

1. Maurício Fiorenza e Diego Kreutz. "Firewalls em Redes Definidas por Software: Estado da Arte." *Anais da XVII Escola Regional de Redes de Computadores*. ERRC, 2019.

2. Maurício Fiorenza, Diego Kreutz, e Rodrigo Mansilha. "Gerenciamento de Firewalls em Redes Híbridas." *Anais do XX Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*. SBSeg, 2020. (*prêmio de melhor artigo curto*)

3. Maurício Fiorenza, Diego Kreutz, Rodrigo Mansilha, Douglas Macedo, Eduardo Feitosa e Roger Immich. "Representação e Aplicação de Políticas de Segurança em Redes Híbridas" *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. SBRC, 2021. (Previsto para submissão em breve).

REFERÊNCIAS

ABDALLAH, S. et al. Fuzzy decision system for technology choice in hybrid networks. In: IEEE. **Fourth International Conference on Software Defined Systems (SDS)**. [S.l.], 2017. p. 106–111.

Adão, P. et al. Mignis: A semantic based tool for firewall configuration. In: IEEE. **IEEE 27th Computer Security Foundations Symposium**. [S.l.], 2014. p. 351–365.

AL-SHAER, E. et al. Conflict classification and analysis of distributed firewall policies. **IEEE J.Sel. A. Commun.**, IEEE Press, v. 23, n. 10, p. 2069–2084, set. 2006. ISSN 0733-8716. Disponível em: https://doi.org/10.1109/JSAC.2005.854119.

Al-Shaer, E. S.; Hamed, H. H. Management and translation of filtering security policies. In: **IEEE International Conference on Communications. ICC.** [S.l.: s.n.], 2003. v. 1, p. 256–260 vol.1.

AL-SHAER, E. S.; HAMED, H. H. Modeling and management of firewall policies. **IEEE Trans. on Netw. and Serv. Manag.**, IEEE Press, v. 1, n. 1, p. 2–10, abr. 2004. ISSN 1932-4537. Disponível em: https://doi.org/10.1109/TNSM.2004.4623689.

BARTAL, Y. et al. Firmato: A novel firewall management toolkit. **ACM Trans. Comput. Syst.**, Association for Computing Machinery, New York, NY, USA, v. 22, p. 381–420, nov. 2004. ISSN 0734-2071. Disponível em: https://doi.org/10.1145/1035582.1035583.

BASET, S. et al. Usable declarative configuration specification and validation for applications, systems, and cloud. In: **Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Industrial Track**. New York, NY, USA: Association for Computing Machinery, 2017. (Middleware '17), p. 29–35. ISBN 9781450352000. Disponível em: https://doi.org/10.1145/3154448.3154453.

BODEI, C. et al. Language-independent synthesis of firewall policies. In: IEEE. **IEEE European Symposium on Security and Privacy (EuroS&P)**. [S.l.], 2018. p. 92–106.

BOSSHART, P. et al. P4: Programming protocol-independent packet processors. **SIGCOMM Comput. Commun. Rev.**, Association for Computing Machinery, New York, NY, USA, v. 44, n. 3, p. 87–95, jul. 2014. ISSN 0146-4833. Disponível em: https://doi.org/10.1145/2656877.2656890.

BOTTA, D. et al. Towards understanding it security professionals and their tools. In: **Proceedings of the 3rd Symposium on Usable Privacy and Security**. New York, NY, USA: Association for Computing Machinery, 2007. (SOUPS '07), p. 100–111. ISBN 9781595938015. Disponível em: https://doi.org/10.1145/1280680.1280693.

CAPROLU, M.; RAPONI, S.; PIETRO, R. D. Fortress: an efficient and distributed firewall for stateful data plane sdn. **Security and Communication Networks**, Hindawi, 2019.

Check Point. **What is URL Filtering?** 2020. <https://www.checkpoint.com/cyber-hub/network-security/what-is-url-filtering/>. Acesso em: 10 jan. 2021.

Cheng, Q. et al. Guarding the perimeter of cloud-based enterprise networks: An intelligent sdn firewall. In: IEEE. **IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)**. [S.l.], 2018. p. 897–902.

Cisco Systems. **A Rede Baseada em Intenção**. 2017. <https://www.cisco.com/c/dam/global/pt_br/solutions/pdfs/intent-networking-wp-cte-pt-br.pdf>. Acesso em: 28 dez. 2020.

Cisco Systems. **Cisco Adaptive Security Device Manager (ASDM)**. 2018. <https://www.cisco.com/c/en/us/products/security/adaptive-security-device-manager/index.html>. Acesso em: 21 jan. 2021.

Cisco Systems. **Firepower System User Management**. 2019. <https://www.cisco.com/c/en/us/td/docs/security/firepower/60/configuration/guide/fpmc-config-guide-v60/Firepower_System_User_Management.html>. Acesso em: 05 jan. 2021.

Cisco Systems. **CLI Book 1: Cisco ASA Series General Operations CLI Configuration Guide**. 2020. <https://www.cisco.com/c/en/us/td/docs/security/asa/asa97/configuration/general/asa-97-general-config/route-static.html>. Acesso em: 10 jan. 2021.

Cisco Systems. **Cisco Intent-Based Networking (IBN)**. 2021. <https://www.cisco.com/c/en/us/solutions/intent-based-networking.html>. Acesso em: 24 fev. 2021.

CLEMM, A. et al. **Intent-Based Networking - Concepts and Overview**. [S.l.], 2019. Work in Progress. Disponível em: https://datatracker.ietf.org/doc/html/draft-clemm-nmrg-dist-intent-03.

CLEMM, A. et al. **Intent-Based Networking - Concepts and Definitions**. [S.l.], 2021. Work in Progress. Disponível em: https://datatracker.ietf.org/doc/html/draft-irtf-nmrg-ibn-concepts-definitions-03.

COONEY, M. **Juniper reinforces its intent-based networking with Apstra buy**. 2020. <https://www.networkworld.com/article/3600127/juniper-reinforces-its-intent-based-networking-with-apstra-buy.html>. Acesso em: 24 fev. 2021.

Cordova, R. F.; Marcovich, A. L.; Santivanez, C. A. An efficient method for ontology-based multi-vendor firewall misconfiguration detection: A real-case study. In: IEEE. **IEEE ANDESCON**. [S.l.], 2018. p. 1–3.

DATTA, R. et al. P4Guard: Designing P4 based firewall. In: IEEE. **IEEE Military Communications Conference (MILCOM)**. [S.l.], 2018. p. 1–6.

FIESSLER, A. et al. FireFlow-high performance hybrid SDN-firewalls with OpenFlow. In: IEEE. **IEEE 43rd Conference on Local Computer Networks (LCN)**. [S.l.], 2018. p. 267–270.

Firewalld. **Rich Language Documentation**. 2021. <https://firewalld.org/documentation/man-pages/firewalld.richlanguage.html>. Acesso em: 22 jan. 2021.

Fortinet. **FortiManager Automation-Driven Network Management**. 2021. <https://www.fortinet.com/products/management/fortimanager>. Acesso em: 24 fev. 2021.

GAO, K.; CONTRERAS, L. M.; RANDRIAMASY, S. Bi-directional network and application interaction: Application intents upon abstracted network information (invited paper). In: **Proceedings of the Workshop on Network Application Integration/CoDesign**. New York, NY, USA: Association for Computing Machinery, 2020. (NAI '20), p. 43–50. ISBN 9781450380447. Disponível em: https://doi.org/10.1145/3405672.3409491.

Gartner. **Technology Insight for Network Security Policy Management**. 2019. <https://www.gartner.com/en/documents/3902564/technology-insight-for-network-security-policy-managemen>. Acesso em: 08 fev. 2021.

Gartner. **Network Firewalls Reviews and Ratings**. 2021. <https://www.gartner.com/reviews/market/network-firewalls>. Acesso em: 17 fev. 2021.

Gufw Project. **Gufw Firewall**. 2021. <http://gufw.org/>. Acesso em: 22 jan. 2021.

HABER, E. M.; BAILEY, J. Design guidelines for system administration tools developed through ethnographic field studies. In: **Proceedings of the 2007 Symposium on Computer Human Interaction for the Management of Information Technology**. New York, NY, USA: Association for Computing Machinery, 2007. (CHIMIT '07), p. 1–es. ISBN 9781595936356. Disponível em: https://doi.org/10.1145/1234772.1234774.

HEORHIADI, V. et al. Intent-driven composition of resource-management sdn applications. In: **Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies**. New York, NY, USA: Association for Computing Machinery, 2018. (CoNEXT '18), p. 86–97. ISBN 9781450360807. Disponível em: https://doi.org/10.1145/3281411.3281431.

HU, H.; AHN, G.-J.; KULKARNI, K. Detecting and resolving firewall policy anomalies. **IEEE Transactions on dependable and secure computing**, IEEE, v. 9, n. 3, p. 318–331, 2012.

HU, H. et al. Flowguard: Building robust firewalls for software-defined networks. In: **Proceedings of the Third Workshop on Hot Topics in Software Defined Networking**. New York, NY, USA: Association for Computing Machinery, 2014. (HotSDN '14), p. 97–102. ISBN 9781450329897. Disponível em: https://doi.org/10.1145/2620728.2620749.

HU, H. et al. Towards a reliable firewall for software-defined networks. **Computers & Security**, v. 87, p. 101597, 2019. ISSN 0167-4048. Disponível em: https://www.sciencedirect.com/science/article/pii/S016740481930152X.

Huawei Technologies. **Huawei Intent-Driven Network White Paper**. 2019. <https://carrier.huawei.com/~/media/CNBG/Downloads/Spotlight/all-cloud-network-towards-5g/idn-en.pdf>. Acesso em: 24 fev. 2021.

IBM X-Force. **X-Force Threat Intelligence Index**. 2020. <https://www.ibm.com/downloads/cas/DEDOLR3W>. Acesso em: 17 fev. 2021.

JACOBS, A. S. et al. Refining network intents for self-driving networks. In: **Proceedings of the Afternoon Workshop on Self-Driving Networks**. New York, NY, USA: Association for Computing Machinery, 2018. (SelfDN 2018), p. 15–21. ISBN 9781450359146. Disponível em: https://doi.org/10.1145/3229584.3229590.

JTC, I. Information technology-syntactic metalanguage-extended bnf. **ISO/IEC Internationl Standard, vol. 14977: 1996 (E)**, ISO/IEC, 1996.

KREUTZ, D. et al. Software-defined networking: A comprehensive survey. **Proceedings of the IEEE**, v. 103, n. 1, p. 14–76, 2014.

KRIT, S.-d.; HAIMOUD, E. Overview of firewalls: Types and policies: Managing windows embedded firewall programmatically. In: IEEE. **International Conference on Engineering & MIS (ICEMIS)**. [S.l.], 2017. p. 1–7.

LINAKER, J. et al. Guidelines for conducting surveys in software engineering. **Lund University**, 2015.

Liu, A. X.; Gouda, M. G. Diverse firewall design. **IEEE Transactions on Parallel and Distributed Systems**, IEEE, v. 19, n. 9, p. 1237–1251, 2008.

Lobo, J.; Marchi, M.; Provetti, A. Firewall configuration policies for the specification and implementation of private zones. In: IEEE. **IEEE International Symposium on Policies for Distributed Systems and Networks**. [S.l.], 2012. p. 78–85.

Maldonado-Lopez, F. A.; Calle, E.; Donoso, Y. Detection and prevention of firewall-rule conflicts on software-defined networking. In: IEEE. **7th International Workshop on Reliable Networks Design and Modeling (RNDM)**. [S.l.], 2015. p. 259–265.

MANSMANN, F.; GöBEL, T.; CHESWICK, W. Visual analysis of complex firewall configurations. In: **Proceedings of the Ninth International Symposium on Visualization for Cyber Security**. New York, NY, USA: Association for Computing Machinery, 2012. (VizSec '12), p. 1–8. ISBN 9781450314138. Disponível em: https://doi.org/10.1145/2379690.2379691.

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

MELO, A. M. et al. Estratégias remotas à avaliação de interfaces de usuário. In: SBC. **Anais da IV Escola Regional de Engenharia de Software**. [S.l.], 2020. p. 245–254.

MORZHOV, S.; ALEKSEEV, I.; NIKITINSKIY, M. Firewall application for Floodlight SDN controller. In: IEEE. **International Siberian Conference on Control and Communications**. [S.l.], 2016. p. 1–5.

MULLER, M. J.; HASLWANTER, J. H.; DAYTON, T. Participatory practices in the software lifecycle. In: **Handbook of human-computer interaction**. [S.l.]: Elsevier, 1997. p. 255–297.

NAGENDRA, V. et al. An intent-based automation framework for securing dynamic consumer iot infrastructures. In: **Proceedings of The Web Conference**. New York, NY, USA: Association for Computing Machinery, 2020. (WWW '20), p. 1625–1636. ISBN 9781450370233. Disponível em: https://doi.org/10.1145/3366423.3380234.

Netfilter. **Documentation about the netfilter/iptables project**. 2021. <http://www.netfilter.org/documentation/>. Acesso em: 22 jan. 2021.

NIELSEN, J. **User Testing: Why & How**. 2018. <https://www.youtube.com/watch?v=v8JJrDvQDF4>.

Oracle. **The Business Impacts of the Modern Data Breach**. 2020. <https://www.oracle.com/a/ocom/docs/cloud/oracle-ctr-2020-impacts-of-data-breach.pdf>. Acesso em: 17 fev. 2021.

OTHMAN, W. M. et al. Implementation and performance analysis of SDN firewall on pox controller. In: IEEE. **IEEE 9th International Conference on Communication Software and Networks (ICCSN)**. [S.l.], 2017. p. 1461–1466.

Palo Alto Networks. **QoS Overview**. 2020. <https://docs.paloaltonetworks.com/pan-os/10-0/pan-os-admin/quality-of-service/qos-overview.html>. Acesso em: 10 jan. 2021.

Palo Alto Networks. **Administrative Role Types**. 2021. <https://docs.paloaltonetworks.com/pan-os/8-1/pan-os-admin/firewall-administration/manage-firewall-administrators/administrative-role-types>. Acesso em: 05 jan. 2021.

pfSense. **The pfSense Documentation**. 2020. <https://docs.netgate.com/manuals/pfsense/en/latest/the-pfsense-documentation.pdf>. Acesso em: 22 jan. 2021.

POZO, S.; VARELA-VACA, A.; GASCA, R. AFPL2, an abstract language for firewall ACLs with NAT support. In: IEEE. **Second International Conference on Dependability**. [S.l.], 2009. p. 52–59.

RAMOS, F. M. V.; KREUTZ, D.; VERISSIMO, P. Software-defined networks: On the road to the softwarization of networking. **Cutter IT Journal**, v. 28, p. 6–13, 05 2015.

RIFTADI, M.; KUIPERS, F. P4i/o: Intent-based networking with P4. In: IEEE. **IEEE Conference on NetSoft**. [S.l.], 2019. p. 438–443.

SANDHYA; SINHA, Y.; HARIBABU, K. A survey: Hybrid sdn. **Journal of Network and Computer Applications**, v. 100, p. 35–55, 2017. ISSN 1084-8045. Disponível em: https://www.sciencedirect.com/science/article/pii/S108480451730317X.

SANVITO, D. et al. Onos intent monitor and reroute service: enabling plug&play routing logic. In: IEEE. **4th IEEE Conference on Network Softwarization and Workshops (NetSoft)**. [S.l.], 2018. p. 272–276.

SAâDAOUI, A.; Ben Youssef Ben Souayeh, N.; BOUHOULA, A. Fare: Fdd-based firewall anomalies resolution tool. **Journal of Computational Science**, v. 23, p. 181–191, 2017. ISSN 1877-7503. Disponível em: https://www.sciencedirect.com/science/article/pii/S1877750317309894.

SCARFONE, K.; HOFFMAN, P. Guidelines on firewalls and firewall policy. **NIST Special Publication**, v. 800, p. 41, 2009.

SCHEID, E. J. et al. A controlled natural language to support intent-based blockchain selection. In: IEEE. **IEEE International Conference on Blockchain and Cryptocurrency (ICBC)**. [S.l.], 2020. p. 1–9.

SINGH, A.; AUJLA, G. S.; BALI, R. S. Intent-based network for data dissemination in software-defined vehicular edge computing. **IEEE Transactions on Intelligent Transportation Systems**, IEEE, p. 1–9, 2020.

SONG, H. Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane. In: **Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking**. New York, NY, USA: Association for Computing Machinery, 2013. (HotSDN '13), p. 127–132. ISBN 9781450321785. Disponível em: https://doi.org/10.1145/2491185.2491190.

SOULE, R. et al. Merlin: A language for managing network resources. **IEEE/ACM Trans. Netw.**, IEEE Press, v. 26, n. 5, p. 2188–2201, out. 2018. ISSN 1063-6692. Disponível em: https://doi.org/10.1109/TNET.2018.2867239.

SUBRAMANYA, T.; RIGGIO, R.; RASHEED, T. Intent-based mobile backhauling for 5g networks. In: IEEE. **12th International Conference on Network and Service Management (CNSM)**. [S.l.], 2016. p. 348–352.

SUN, Q.; LIU, W. S.; XIE, K. **An Intent-driven Management Framework**. [S.l.], 2019. Work in Progress. Disponível em: https://datatracker.ietf.org/doc/html/draft-sun-nmrg-intent-framework-00.

TIAN, B. et al. Safely and automatically updating in-network acl configurations with intent language. In: **Proceedings of the ACM Special Interest Group on Data Communication**. New York, NY, USA: Association for Computing Machinery, 2019, (SIGCOMM '19). p. 214–226. ISBN 9781450359566. Disponível em: https://doi.org/10.1145/3341302.3342088.

TRAN, T.; AL-SHAER, E.; BOUTABA, R. Policyvis: Firewall security policy visualization and inspection. In: **Proceedings of the 21st Conference on Large Installation System Administration Conference**. USA: USENIX Association, 2007. (LISA'07). ISBN 9781593271527.

Vinh Tran, T.; Ahn, H. Flowtracker: A sdn stateful firewall solution with adaptive connection tracking and minimized controller processing. In: **International Conference on Software Networking (ICSN)**. [S.l.: s.n.], 2016. p. 1–5.

Visoottiviseth, V. et al. Reflo: Reactive firewall system with openflow and flow monitoring system. In: IEEE. **TENCON - IEEE Region 10 Conference**. [S.l.], 2017. p. 2273–2278.

VISSICCHIO, S.; VANBEVER, L.; BONAVENTURE, O. Opportunities and research challenges of hybrid software defined networks. **SIGCOMM Comput. Commun. Rev.**, Association for Computing Machinery, New York, NY, USA, v. 44, n. 2, p. 70–75, abr. 2014. ISSN 0146-4833. Disponível em: https://doi.org/10.1145/2602204.2602216.

VMware. **What is intent-based networking (IBN)?** 2020. <https://www.vmware.com/topics/glossary/content/intent-based-networking>. Acesso em: 10 fev. 2021.

VORONKOV, A. et al. Systematic literature review on usability of firewall configuration. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 50, n. 6, dez. 2017. ISSN 0360-0300. Disponível em: https://doi.org/10.1145/3130876.

VORONKOV, A.; MARTUCCI, L. A. Natural vs. technical language preference and their impact on firewall configuration. In: SPRINGER. **International Conference on Human-Computer Interaction**. [S.l.], 2020. p. 261–270.

VORONKOV, A.; MARTUCCI, L. A.; LINDSKOG, S. System administrators prefer command line interfaces, don't they? an exploratory study of firewall interfaces. In: **Fifteenth Symposium on Usable Privacy and Security (SOUPS)**. Santa Clara, CA: USENIX Association, 2019. Disponível em: https://www.usenix.org/conference/soups2019/presentation/voronkov.

VÖRÖS, P.; KISS, A. Security middleware programming using p4. In: TRYFONAS, T. (Ed.). **Human Aspects of Information Security, Privacy, and Trust**. Cham: Springer International Publishing, 2016. p. 277–287. ISBN 978-3-319-39381-0.

WEI, Y.; PENG, M.; LIU, Y. Intent-based networks for 6g: Insights and challenges. **Digital Communications and Networks**, v. 6, n. 3, p. 270–280, 2020. ISSN 2352-8648. Disponível em: https://www.sciencedirect.com/science/article/pii/S2352864820302418.

WONG, T. On the usability of firewall configuration. In: **Symposium on usable privacy and security**. [S.l.: s.n.], 2008.

YANG, L. et al. **Forwarding and Control Element Separation (ForCES) Framework**. RFC Editor, 2004. RFC 3746. (Request for Comments, 3746). Disponível em: https://rfc-editor.org/rfc/rfc3746.txt.

Zeydan, E.; Turk, Y. Recent advances in intent-based networking: A survey. In: IEEE. **IEEE 91st Vehicular Technology Conference (VTC-Spring)**. [S.l.], 2020. p. 1–5.

ZHANG, B. et al. Specifications of a high-level conflict-free firewall policy language for multi-domain networks. In: **Proceedings of the 12th ACM Symposium on Access Control Models and Technologies**. New York, NY, USA: Association for Computing Machinery, 2007. (SACMAT '07), p. 185–194. ISBN 9781595937452. Disponível em: https://doi.org/10.1145/1266840.1266871.

ZKIK, K.; HAJJI, S. E.; ORHANOU, G. Design and implementation of a new security planee for hybrid distributed sdns. **Journal of communications.**, v. 14, n. 1, p. 26–32, 2019.

APÊNDICE A – MARCADORES DA GRAMÁTICA FWLANG

A seguir são detalhadas expressões e marcadores que compõem a linguagem FWlang. A gramática, em padrão ENBF, pode ser vista na Figura 5.

- **define intent *policy_type***: define o início da declaração de uma intenção. O termo *policy_type* deve ser substituído pelo tipo de intenção que será aplicada (*e.g.*, acl, *url_filter traffic_shaping*, nat_1to1, nat_nto1, *route*).

- **from** e **to**: definem a origem e o destino dos pacotes tratados pela intenção. Estes marcadores fazem parte dos seis tipos de políticas mapeadas na Tabela 4. A utilização destes marcadores é vinculada aos marcadores auxiliares, que definem os tipos de origens e destinos:

  - **endpoint**: para especificar um nó único na rede, que pode ser um *host* ou outro dispositivo qualquer presente na rede. A especificação do nó pode ser realizada através de endereços IP, nomes, ou ainda o valor pré-definido "*any*", para representar qualquer origem ou destino.

  - **range**: para especificar blocos de IPs, ou sub-redes. O intervalo de IPs é definido utilizando máscaras de rede, como 10.0.0.0/24 ou 10.0.0.0/255.255.255.0.

  - **category**: é um marcador utilizado para representar categorias de tráfegos. Firewalls de nova geração permitem a criação de regras (*e.g*, filtros de URL ou *traffic shaping*) baseadas em categorias pré-definidas de tráfego, como *social-media* ou *phishing*, por exemplo. Em firewalls modernos, as categorias são pré-definidas e mantidas/atualizadas pelos fabricantes dos equipamentos.

  - **gateway**: é um marcador específico para políticas de roteamento, que determinam o novo *gateway* para um *host* ou uma rede. O marcador permite a utilização de endereços IP, blocos de endereços, ou nome de um *host*.

- **name**: é um marcador para atribuir nomes específicos às políticas criadas. Seu uso é obrigatório para qualquer tipo de intenção. Em intenções dos tipos ACL, *traffic shaping* e filtros de URL, a ordem de prioridade é estabelecida fazendo-se referência ao nome da intenção. Por exemplo, para dizer que a intenção *"acl-allow-https"* deve ser aplicada antes da intenção *"acl-deny-all-traffic"*. O

marcador de nome é utilizado em conjunto com *order*, para definir a ordem de prioridade da regra nos firewalls (e.g., *order before("acl-deny-all-traffic")*. O nome (valor) da intenção é determinado a através do marcador auxiliar **text**, cujos valores definidos são livres (e.g., *name text('acl-allow-https')*).

- **order**: é um marcador obrigatório em regras do tipo ACL, *traffic shaping* e filtros de URL. Este marcador indica a posição da intenção nas tabelas de regras dos firewalls. O marcador **order** é acompanhado pelos marcadores auxiliares *before* e *after*.

  - **before**: indica a política que a nova intenção deve preceder. Por exemplo, "*before('permit-ftp-all')*" indica que a nova regra deve preceder (ordem de prioridade) a regra "*permit-ftp-all*". O valor predefinido "*all-intents*" pode ser utilizado para indicar que a regra deve preceder todas as demais.
  - **after**: indica a política que a nova intenção deve suceder. Por exemplo, "*after(permit-ftp-all)*" indica que a nova regra deve suceder (ordem de prioridade) a regra "*permit-ftp-all*". O valor predefinido "*all-intents*" pode ser utilizado para indicar que a regra deve suceder todas as demais.

- **allow** e **deny**: representam a ação a ser tomada (permitir ou bloquear) sobre o tráfego especificado na intenção. Estes marcadores de ação são utilizados em intenções do tipo ACL ou filtros de URL, e é acompanhado do marcador auxiliar *traffic*:

  - **traffic**: é utilizado para especificar o serviço, protocolo ou porta que a intenção irá tratar. Os valores para este marcadores incluem o *id* do serviço, como HTTP, SMTP (*Simple Mail Transfer Protocol*), e FTP, a definição do protocolo e porta (*e.g*, TCP/389, UDP/5555) ou o valor "*any*", para qualquer tipo de tráfego.

- **for**: é um marcador utilizado para definição de tráfegos em políticas que não utilizam ações de liberação ou bloqueio (*allow*/*deny*). Em políticas de *traffic shaping* (vide o exemplo da Figura 6(b)) e filtros de URL, a utilização do marcador deve acompanhar o marcador auxiliar **traffic**, como definido no item anterior. Em políticas do tipo NAT 1to1, o uso do **for** é realizado em conjunto com o marcador **port**.

  - **port**: é utilizado na representação de políticas NAT 1to1 que demandem informar o protocolo e as portas de origem e destino no processo de translado do IP público para o IP interno. A sintaxe para composição do valor deste marcador segue o padrão: "protocol: version ; src_port: number ; dst_port: number", onde "version" representa o protocolo (*e.g*, TCP ou UDP), e "number" representam as portas de origem e destino. Um exemplo de intenção deste tipo pode ser visualizado na Figura 18.

- **with**: é um marcador utilizado para definição de largura de banda em políticas de *traffic shaping* e filtro de URL. O marcador é acompanhado pelo marcador auxiliar **throughput**.

  - **throughput**: define a largura de banda atribuída à intenção. O valor pode ser definido utilizando medidas como Kbps, Mbps e Gbps (*e.g.*, 100Mbps).

- **add** e **del**: são marcadores utilizados em todas as políticas. Estes marcadores definem a ação a ser tomada em relação à intenção. O **add** indica que intenção deve ser adicionada aos dispositivos de firewall. Intuitivamente, o **del** indica que a intenção deve ser removida dos firewalls. Estes marcadores são acompanhados de um marcador adicional (e.g., **firewall** ou **middlebox**) para identificar os dispositivos de firewall:

  - **middlebox** e **firewall**: são marcadores equivalentes, que definem os firewall (ou *appliance*) que deverá receber a regra gerada a partir da intenção. Os valores utilizados neste marcador (*e.g*, "*firewall(firewall_id)*") são oriundos da solução de tradução/aplicação de políticas que utiliza a FWlang, ou seja, são a identificação definida para o firewall na solução. Um exemplo disso, são os alias definidos para os firewalls na implementação do FWunify, conforme pode ser visto na Figura 20. Estes marcadores podem ser utilizados para indicar um único firewall (*e.g*, "*add firewall(cisco-1)")*, ou listas de firewalls, separando-os com vírgulas (*e.g*, "*add firewalls(cisco-1,iptables-1,openflow-1)*").

- **start** e **end**: são marcadores utilizados para o intervalo de tempo de atividade da intenção. Para especificar este intervalo, são utilizados os seguintes marcadores auxiliares:

  - **hour**: para indicar a hora e o minuto de início e fim do período de vigência da regra, seguindo o formato de 24 horas "HH:MM" (*i.e.*, 00:00 a 23:59).
  - **date**: para indicar o dia de início e fim do período de vigência da regra. Neste caso, a regra começa a valer às 00:00 da data de início, até às 23:59 da data de fim. O formato de data utilizado no valor deste marcador é "DD/MM/AAAA" (*i.e*, 15/02/2020).
  - **datatime**: para indicar a data e hora de início e fim do período de vigência da regra. O formato de data utilizado no valor deste marcador é "DD/MM/AAAA-HH:MM" (*i.e*, 15/02/2020-12:00)

- **log**: é um marcador que especifica a ativação ou não da geração de registros (*logs*) de atividade da intenção. Quando o marcador é suprimido na descrição da intenção, o valor padrão é não gerar *logs*, isto é, o serviço de registros é desativado. O marcador **log** é utilizado em conjunto com os marcadores auxiliares:
  - **enable**: que indica a ativação do serviço de *logs*.
  - **disable** que indica a desativação do serviço de *logs*.
- **description**: é um marcador opcional que permite a inclusão de uma descrição textual na especificação da intenção. O marcador é acompanhado pelo marcador auxiliar **text** (*e.g.*, "*description text(esta intenção libera o acesso ao servidor do banco de dados MySQL aos servidores de aplicação Web www-app1 e www-app2)*").

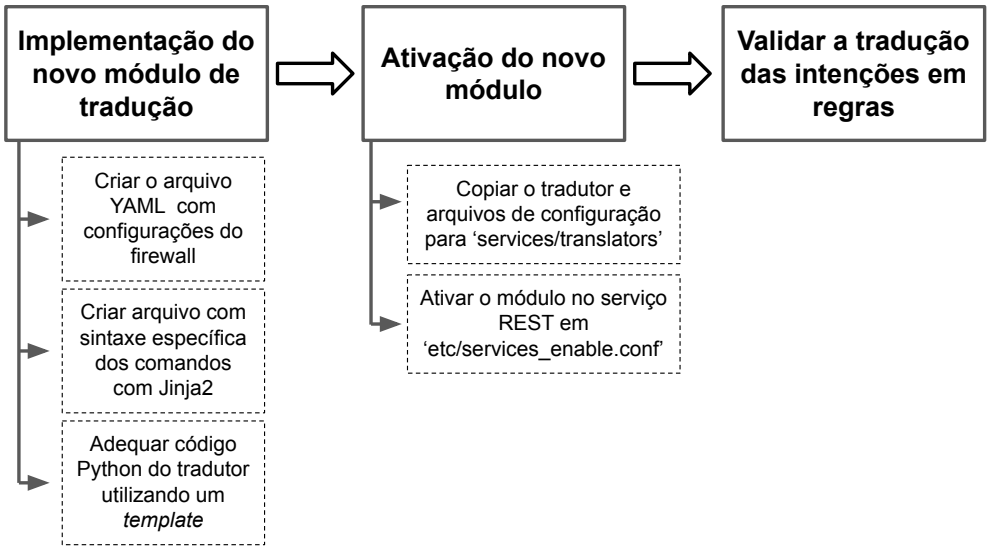
Figura 18 – Intenção de NAT 1to1 em FWlang

```
define intent nat_1to1:
name      text(nat-200.19.0.2-10.0.0.2)
from      endpoint(200.19.0.2)
to        endpoint(10.0.0.2)
for       port(protocol:tcp;src_port:90;dst_port:80)
add       firewall(iptables-1)
```

APÊNDICE B – PASSO-A-PASSO: ADICIONANDO UM NOVO TRADUTOR

A Figura 19 resume o processo de inclusão de novos módulos de tradução (microsserviços de tradução) ao FWunify. Um novo módulo pode ser adicionado para atender especificidades de atualizações de software de alguns firewalls (*e.g.*, um dos firewalls Cisco ASA foi atualizado da versão 8.2 para a versão 8.4) da rede ou a inclusão de um novo tipo de firewall. O processo de adição de um novo tradutor pode ser dividido em duas etapas: (i) a implementação do módulo a partir de *templates* existentes, e (ii) a ativação do módulo na arquitetura.

Figura 19 – Processo de adição de um novo tradutor ao FWunify



Os módulos de tradução são compostos por três arquivos[1]: (a) um arquivo de configuração YAML (*Yet Another Markup Language*)[2] com as informações do firewall a ser gerenciado, como IP de gerência, credenciais de acesso (*e.g.*, usuário e senha), e interfaces (*e.g*, interna, externa, DMZ (*Demilitarized Zone*)); (b) um arquivo *template*, em linguagem Jinga2, que especifica as partes estáticas e dinâmicas da sintaxe (necessária para a tradução) dos comandos do firewall, onde as partes dinâmicas serão preenchidas com as informações vindas da intenção; e (c) um arquivo com o código Python que implementa a tradução das informações vindas da intenção em comandos específicos ao firewall. O código do tradutor recebe um dicionário Python com as informações extraídas da intenção pelo serviço REST, processa as traduções específicas (*e.g.*, mudar

[1]Exemplos de módulos podem ser vistos em <https://github.com/mmfiorenza/fwunify>

[2]YAML é uma linguagem conhecida pelos administradores de sistemas pelo fato de ser utilizada na configuração de diversos serviços em sistemas Linux.

uma máscara de rede para o padrão CIDR, ou converter uma largura de banda de Mbps para Kbps) e utiliza o *template* Jinja2 para compor os comandos da regra na sintaxe específica do firewall.

O novo tradutor deve ser copiado para a pasta "services/translators" do estrutura de arquivos da FWunify. Para habilitar o novo microsserviço de tradução, o serviço REST deve ser informado da existência do novo módulo de tradução. Isso é realizado através do arquivo de configuração "etc/services_enable.conf", onde é especificado o identificador (que é utilizado nas intenções) do novo firewall, além da função da chamada remota definida na implementação do tradutor (*e.g.*, *rpc_cisco_asa_5505_translator.translate_intent*). A Figura 20 apresenta um exemplo de conteúdo deste arquivo de configuração. A partir deste ponto, as intenções podem ser enviadas ao novo tradutor, utilizando o identificador (*alias*) do firewall nos marcadores *middlebox(es)* ou *firewall(s)* da intenção.

Figura 20 – Arquivo de exemplo "etc/services_enable.conf"

```
#alias       #function
cisco-1     cisco_translator.translate_intent
iptables-1  iptables_translator.translate_intent
openflow-1  openflow_translator.translate_intent
paloalto-1  paloalto_translator.translate_intent
```

Para aplicação das regras no firewall, o tradutor utiliza também um dos componentes disponíveis na camada da interface sul. Supondo que o firewall suporte acesso remoto via SSH, basta utilizar o conector SSH disponível. Caso contrário, poderá ser necessário implementar um novo conector na camada da interface sul (*e.g.*, um conector NetConf). A identificação do conector a ser utilizado deve ser feita no código do tradutor, onde deve ser indicada a chamada remota definida na implementação do tradutor.

APÊNDICE C – TRADUÇÃO DE INTENÇÕES FWLANG

As Figuras 21, 22 e 23 apresentam a tradução das intenções FWunify discutidas na Seção 6.4. A Figura 21 apresenta a intenção do tipo ACL, as sintaxes esperadas para firewalls IPTables e OpenFlow, e os comandos gerados através da FWunify. As Figuras 22 e 23 apresentam as intenções, sintaxes esperadas e comandos gerados para as políticas de NAT 1to1 e *traffic shaping*. Como podemos observar, as traduções para IPTables e OpenFlow geram comandos 100% compatíveis com as sintaxes esperadas. É importante ressaltar que o IPTables não possui comandos nativos para *traffic shaping* (Figura 23).

Figura 21 – ACL em FWlang e comandos IPTables e OpenFlow

Intenção
*define intent **acl**:*
*name text(**rule-acl-1**)*
*from range(**10.0.0.0/24**)*
*to range(**200.19.0.0/24**)*
*order before(**all-intents**)*
*deny traffic(**icmp**)*
*add firewall(**cisco-1**,**iptables-1**,**openflow-1**)*

Sintaxe do comando esperado IPTables:
*iptables -I "**chain**" "**position**" -s "**IP/rede de origem**"-d "**IP/rede de origem**" -p "**protocolo**" -j "**permitir/bloquear**"*

Comando gerado IPTables:
*iptables -I **FORWARD 1** -s **10.0.0.0/255.255.255.0** -d **200.19.0.0/255.255.255.0** -p **icmp** -j **DROP***

Sintaxe do comando esperado OpenFlow:
*ovs-ofctl add-flow "**switch**" dl_type=0x800,priority="**posição**",nw_src="**IP/rede de origem**",nw_dst="**IP/rede de destino**", nw_proto="**nº protocolo IANA**",icmp_type="**tipo ICMP IANA**",action="**permitir/bloquear**"*

Comando gerado OpenFlow:
*ovs-ofctl add-flow **s1** dl_type=0x800,priority=**65535**,nw_src=**10.0.0.0/255.255.255.0**,nw_dst=**200.19.0.0/255.255.255.0**, nw_proto=**1**,icmp_type=**8**,action=**drop***

Figura 22 – NAT 1to1 em FWlang e comandos IPTables e OpenFlow

Intenção
*define intent **nat_1to1**:*
*name text(**nat-200.19.0.50-10.0.0.50**)*
*from endpoint(**200.19.0.50**)*
*to endpoint(**10.0.0.50**)*
*for port(protocol:**tcp**;scr_port:**80**;dst_port:**90**)*
*del firewall(**cisco-1**,**iptables-1**,**openflow-1**)*

Sintaxe do comando esperado IPTables:
*iptables -t nat -D PREROUTING -d "**IP válido**" -p "**protocolo**" --dport "**porta origem**" -j DNAT --to-destination "**IP interno**":"**porta destino**"*
*iptables -t nat -D POSTROUTING -s "**IP interno**" -p "**protocolo**" --sport "**porta destino**" -j SNAT --to-source "**IP válido**":"**porta origem**"*

Comando gerado IPTables:
*iptables -t nat -D PREROUTING -d **200.19.0.50** -p **tcp** --dport **80** -j DNAT --to-destination **10.0.0.50:90***
*iptables -t nat -D POSTROUTING -s **10.0.0.50** -p tcp --sport **90** -j SNAT --to-source **200.19.0.50:80***

Sintaxe do comando esperado OpenFlow:
*ovs-ofctl del-flows "**switch**" "**protocolo**",nw_dst="**IP valido**",tp_dst="**porta origem**"*
*ovs-ofctl del-flows "**switch**" "**protocolo**",nw_src="**IP interno**",tp_src="**porta destino**"*

Comando gerado OpenFlow:
*ovs-ofctl del-flows **s1 tcp**,nw_dst=**200.19.0.50**,tp_dst=**80***
*ovs-ofctl del-flows **s1 tcp**,nw_src=**10.0.0.50**,tp_src=**90***

Figura 23 – *Traffic shaping* em FWlang e comandos OpenFlow

Intenção em FWlang:

define intent ***traffic_shaping****:*

name	*text(**rule-ts-1**)*
from	*range(**10.0.0.0/24**)*
to	*endpoint(**200.19.0.100**)*
order	*before(**all-intents**)*
for	*traffic(**ftp**)*
with	*throughput(**30Mbps**)*
add	*firewall(**cisco-1**,**openflow-1**)*

Sintaxe do comando esperado OpenFlow:

*ovs-vsctl set interface "**switch**"-"**interface**" ingress_policing_rate="**largura de banda**" ingress_policing_burst="**burst size**"*

Comando gerado OpenFlow:

*ovs-vsctl set interface **s1-eth0** ingress_policing_rate=**30000** ingress_policing_burst=**3000***

APÊNDICE D – APLICAÇÃO DE INTENÇÕES ACL

A Figura 24 apresenta uma intenção ACL em FWlang. Esta intenção é utilizada para adicionar (*add*) e remover (*del*) uma regra que bloqueia o tráfego HTTP entre a rede dos *hosts* 1, 2 e 3 (10.0.0.0/24) e o *server-1* (IP 200.19.0.100) do ambiente de testes descrito na Seção 6.1. As Figuras 25 e 26 apresentam o resultado do processamento da intenção de ACL utilizando a FWunify. No caso, a URL *http://localhost:5000* identifica a localização do serviço REST da FWunify. Como pode ser observado nas figuras, o envio da intenção de ACL, especificada nos respectivos arquivos de texto (".txt"), é realizada através do comando *curl*. Para enviar a intenção, é necessário definir as credenciais (*i.e.*, login e senha) do operador, o arquivo que contém a intenção em FWlang (e.g., "*deny-net-server-http-add.txt*") e o endereço do serviço REST, para onde a intenção é enviada.

Figura 24 – Intenção ACL descrita com FWlang

```
define intent acl:
name        text(deny-net-server-http)
from        range(10.0.0.0/24)
to          endpoint(200.19.0.100)
deny        traffic(http)
order       before(all-intents)
add/del     firewall(cisco-1,iptables-1,openflow-1)
```

Figura 25 – FWunify: inserção da intenção nos firewalls

```
root@prototype:~# curl -u user1:user1 --data-binary "@deny-net-server-http-add.txt" http://localhost:5000

--> Return of module CISCO-1
---------------------------------
configure terminal
!
access-list inside_access_in line 1 extended deny tcp 10.0.0.0 255.255.255.0 host 200.19.0.100 eq 80
access-group inside_access_in in interface inside
clear conn
!
exit


Commands applied in the firewall CISCO-1: OK
---------------------------------
--> Return of module IPTABLES-1
---------------------------------

iptables -I FORWARD 1 -s 10.0.0.0/255.255.255.0 -d 200.19.0.100 -p tcp --dport 80 -j DROP


Commands applied in the firewall IPTABLES-1: OK
---------------------------------
--> Return of module OPENFLOW-1
---------------------------------

ovs-ofctl add-flow s1 dl_type=0x800,priority=65535,nw_src=10.0.0.0/255.255.255.0,nw_dst=200.19.0.100,nw_proto=6
,tcp_dst=80,action=drop


Commands applied in the firewall OPENFLOW-1: OK
---------------------------------
```

Figura 26 – FWunify: remoção da intenção nos firewalls

```
root@prototype:~# curl -u user1:user1 --data-binary "@deny-net-server-http-del.txt" http://localhost:5000

--> Return of module CISCO-1
---------------------------------
configure terminal
!
no access-list inside_access_in extended deny tcp 10.0.0.0 255.255.255.0 host 200.19.0.100 eq 80
clear conn
!
exit


Commands applied in the firewall CISCO-1: OK
---------------------------------
--> Return of module IPTABLES-1
---------------------------------

iptables -D FORWARD -s 10.0.0.0/255.255.255.0 -d 200.19.0.100 -p tcp --dport 80 -j DROP


Commands applied in the firewall IPTABLES-1: OK
---------------------------------
--> Return of module OPENFLOW-1
---------------------------------

ovs-ofctl del-flows s1 dl_type=0x800,nw_src=10.0.0.0/255.255.255.0,nw_dst=200.19.0.100,nw_proto=6,tcp_dst=80,


Commands applied in the firewall OPENFLOW-1: OK
---------------------------------
```

APÊNDICE E – APLICAÇÃO DE INTENÇÃO DE *TRAFFIC SHAPING*

A saída da execução do comando *Iperf* é apresentada na Listagem E.1. Como pode ser observado, a vazão reduz para 98.1Mbps quando é aplicada a intenção de *traffic shaping*. A Figura 27 apresenta a intenção de *traffic shaping* que foi utilizada limitar em 100 Mbps o tráfego (5555/UDP) entre a rede do *host-3* (10.0.0.0/24) e o *server-1* (IP 200.19.0.100). Na Figura 27 são apresentados os comandos e as saídas do processo de inserção e remoção consecutiva da intenção de *traffic shaping* utilizando a FWunify. A intenção é enviada à FWunify através do comando *curl*.

Listagem E.1: *Log* de execução do Iperf

```
user@server-1:~# iperf -u -s -p 5555 -i 5
------------------------------------------------------------
Server listening on UDP port 5555
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ 11] local 200.19.0.100 port 5555 connected with 10.0.0.10 port 44288
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[ 11]  0.0- 5.0 sec   639 MBytes  1.07 Gbits/sec   0.000 ms  668/456545 (0.15%)
[ 11]  5.0-10.0 sec   639 MBytes  1.07 Gbits/sec   0.000 ms  367/456522 (0.08%)
[ 11] 10.0-15.0 sec   640 MBytes  1.07 Gbits/sec   0.000 ms    0/456523 (0%)
[ 11] 15.0-20.0 sec   640 MBytes  1.07 Gbits/sec   0.000 ms  209/456523 (0.046%)
[ 11] 20.0-25.0 sec   281 MBytes   472 Mbits/sec   0.001 ms 255963/456524 (56%)
[ 11] 25.0-30.0 sec  58.5 MBytes  98.1 Mbits/sec   0.001 ms 414807/456530 (91%)
[ 11] 30.0-35.0 sec  58.5 MBytes  98.1 Mbits/sec   0.000 ms 414798/456520 (91%)
[ 11] 35.0-40.0 sec  58.5 MBytes  98.1 Mbits/sec   0.000 ms 414798/456520 (91%)
[ 11] 40.0-45.0 sec  58.5 MBytes  98.1 Mbits/sec   0.000 ms 414807/456530 (91%)
[ 11] 45.0-50.0 sec   386 MBytes   648 Mbits/sec   0.000 ms 180931/456514 (40%)
[ 11] 50.0-55.0 sec   639 MBytes  1.07 Gbits/sec   0.000 ms  656/456523 (0.14%)
[ 11] 55.0-60.0 sec   640 MBytes  1.07 Gbits/sec   0.000 ms    0/456522 (0%)
[ 11] 60.0-65.0 sec   640 MBytes  1.07 Gbits/sec   0.000 ms    0/456523 (0%)
[ 11] 65.0-70.0 sec   639 MBytes  1.07 Gbits/sec   0.001 ms  499/456523 (0.11%)
[ 11] 70.0-75.0 sec   290 MBytes   486 Mbits/sec   0.000 ms 249710/456524 (55%)
[ 11] 75.0-80.0 sec  58.5 MBytes  98.1 Mbits/sec   0.000 ms 414807/456530 (91%)
[ 11] 80.0-85.0 sec  58.5 MBytes  98.1 Mbits/sec   0.000 ms 414798/456520 (91%)
[ 11] 85.0-90.0 sec  58.5 MBytes  98.1 Mbits/sec   0.000 ms 414798/456520 (91%)
[ 11] 90.0-95.0 sec  58.5 MBytes  98.1 Mbits/sec   0.000 ms 414807/456530 (91%)
[ 11] 95.0-100.0 sec   406 MBytes   681 Mbits/sec   0.000 ms 166870/456513 (37%)
[ 11] 100.0-105.0 sec   640 MBytes  1.07 Gbits/sec   0.000 ms  288/456523 (0.063%)
[ 11] 105.0-110.0 sec   640 MBytes  1.07 Gbits/sec   0.000 ms    0/456523 (0%)
[ 11] 110.0-115.0 sec   640 MBytes  1.07 Gbits/sec   0.000 ms  153/456523 (0.034%)
[ 11] 115.0-120.0 sec   640 MBytes  1.07 Gbits/sec   0.000 ms  127/456523 (0.028%)
user@server-1:~#
```

Figura 27 – Intenção de *traffic shaping* descrita com FWlang

```
define intent traffic_shaping:
name                text(limit-net-h100-100m)
from                range(10.0.0.0/24)
to                  endpoint(200.19.0.100)
order               before(all-intents)
for                 traffic(udp/5555)
with                throughput(100Mbps)
add/del             firewall(openflow-1)
```

Figura 28 – Inserção e remoção da intenção de *traffic shaping*

```
root@prototype:~# curl -u user1:user1 --data-binary "@limit-net-h100-100m-add.txt" http://localhost:5000

--> Return of module OPENFLOW-1
---------------------------------

ovs-vsctl set interface s1-eth1 ingress_policing_rate=100000 ingress_policing_burst=10000


Commands applied in the firewall OPENFLOW-1: OK
---------------------------------

root@prototype:~# curl -u user1:user1 --data-binary "@limit-net-h100-100m-del.txt" http://localhost:5000

--> Return of module OPENFLOW-1
---------------------------------

ovs-vsctl set interface s1-eth1 ingress_policing_rate=0


Commands applied in the firewall OPENFLOW-1: OK
---------------------------------

root@prototype:~# curl -u user1:user1 --data-binary "@limit-net-h100-100m-add.txt" http://localhost:5000

--> Return of module OPENFLOW-1
---------------------------------

ovs-vsctl set interface s1-eth1 ingress_policing_rate=100000 ingress_policing_burst=10000


Commands applied in the firewall OPENFLOW-1: OK
---------------------------------

root@prototype:~# curl -u user1:user1 --data-binary "@limit-net-h100-100m-del.txt" http://localhost:5000

--> Return of module OPENFLOW-1
---------------------------------

ovs-vsctl set interface s1-eth1 ingress_policing_rate=0


Commands applied in the firewall OPENFLOW-1: OK
---------------------------------
```

APÊNDICE F – REPRESENTAÇÃO DAS SEIS ACLS EM FWLANG

As intenções da Figura 29 representam as seis políticas do tipo ACL utilizadas na Seção 6.6. As intenções foram enviadas à FWunify sequencialmente. O processamento e aplicação das regras geradas leva em consideração a ordem de prioridade definida na intenção. Por exemplo, a intenção "drop-all-all" deve ser adicionada após todas as demais (*order after(all-intents)*). As duas intenções seguintes, com a ordem de prioridade (*order before(all-intents)*), resultam em regras de maior prioridade no firewall. As regras da intenção "permit-h10-h20-mysql" precedem (*order before(drop-all-all)*) as da intenção "drop-all-all", enquanto a intenção "drop-net-h20-mysql" deve suceder a intenção "permit-h10-h20-mysql". A intenção "drop-incident-h21", apesar de ser a última enviada aos firewalls, possui a maior prioridade nas tabelas de regras dos firewalls (*order before(all-intents)*). O resultado final da aplicação das políticas nos firewalls pode ser visualizado na Figura17 da Seção 6.6 e nas Figuras 31 e 33 do Apêndice G.

Figura 29 – Intenções das ACLs em FWlang

```
define intent acl:
name        text(drop-all-all)
from        endpoint(any)
to          endpoint(any)
deny        traffic(any)
order       after(all-intents)
add         firewall(cisco-1,iptables-1,openflow-1)
```

(a) *Intenção 1*

```
define intent acl:
name        text(permit-net-all-http)
from        range(10.0.0.0/24)
to          endpoint(any)
allow       traffic(http)
order       before(all-intents)
add         firewall(cisco-1,iptables-1,openflow-1)
```

(b) *Intenção 2*

```
define intent acl:
name        text(permit-net-all-https)
from        range(10.0.0.0/24)
to          endpoint(any)
allow       traffic(https)
order       before(all-intents)
add         firewall(cisco-1,iptables-1,openflow-1)
```

(c) *Intenção 3*

```
define intent acl:
name        text(permit-h10-h20-mysql)
from        endpoint(10.0.0.10)
to          endpoint(200.19.0.10)
allow       traffic(tcp/3306)
order       before(drop-all-all)
add         firewall(cisco-1,iptables-1,openflow-1)
```

(d) *Intenção 4*

```
define intent acl:
name        text(deny-net-h20-mysql)
from        range(10.0.0.0/24)
to          endpoint(200.19.0.10)
deny        traffic(tcp/3306)
order       after(permit-h10-h20-mysql)
add         firewall(cisco-1,iptables-1,openflow-1)
```

(e) *Intenção 5*

```
define intent acl:
name        text(deny-incident-h21)
from        endpoint(any)
to          endpoint(200.19.0.20)
deny        traffic(any)
order       before(all-intents)
add         firewall(cisco-1,iptables-1,openflow-1)
```

(f) *Intenção 6*

APÊNDICE G – APLICAÇÃO MANUAL E AUTOMÁTICA DAS SEIS ACLS

As Figuras 30, 31, 32 e 33 apresentam as listas de regras das inserções manuais e automática (via FWunify) das políticas de ACL definidas na Seção 6.6.

Figura 30 – Lista de regras das inserções manuais no IPTables

```
root@firewall-2-iptables:/home/iptables#
root@firewall-2-iptables:/home/iptables# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
DROP       all  --  anywhere             200.19.0.20
ACCEPT     tcp  --  10.0.0.0/24          anywhere             tcp dpt:https
ACCEPT     tcp  --  10.0.0.0/24          anywhere             tcp dpt:http
ACCEPT     tcp  --  10.0.0.10            200.19.0.10          tcp dpt:mysql
DROP       tcp  --  10.0.0.0/24          200.19.0.10          tcp dpt:mysql
DROP       all  --  anywhere             anywhere

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@firewall-2-iptables:/home/iptables#
```

Figura 31 – Lista de regras das via FWunify no IPTables

```
root@firewall-2-iptables:/home/iptables#
root@firewall-2-iptables:/home/iptables# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
DROP       all  --  anywhere             200.19.0.20
ACCEPT     tcp  --  10.0.0.0/24          anywhere             tcp dpt:https
ACCEPT     tcp  --  10.0.0.0/24          anywhere             tcp dpt:http
ACCEPT     tcp  --  10.0.0.10            200.19.0.10          tcp dpt:mysql
DROP       tcp  --  10.0.0.0/24          200.19.0.10          tcp dpt:mysql
DROP       all  --  anywhere             anywhere

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@firewall-2-iptables:/home/iptables#
```

Figura 32 – Lista de regras das inserções manuais no OpenFlow

```
/ #
/ # ovs-ofctl dump-flows br0
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=2.872s, table=0, n_packets=0, n_bytes=0, idle_age=2, priority=32200,ip,nw_dst=200.19.0.20 actions=drop
 cookie=0x0, duration=68.046s, table=0, n_packets=0, n_bytes=0, idle_age=68, priority=32000,tcp,nw_src=10.0.0.0/24,tp_dst=80
actions=NORMAL
 cookie=0x0, duration=59.575s, table=0, n_packets=0, n_bytes=0, idle_age=59, priority=32100,tcp,nw_src=10.0.0.0/24,tp_dst=443
 actions=NORMAL
 cookie=0x0, duration=21.029s, table=0, n_packets=0, n_bytes=0, idle_age=21, priority=31900,tcp,nw_src=10.0.0.10,nw_dst=200.1
9.0.10,tp_dst=3306 actions=NORMAL
 cookie=0x0, duration=9.831s, table=0, n_packets=0, n_bytes=0, idle_age=9, priority=31800,tcp,nw_src=10.0.0.0/24,nw_dst=200.1
9.0.10,tp_dst=3306 actions=drop
 cookie=0x0, duration=87.956s, table=0, n_packets=0, n_bytes=0, idle_age=87, priority=15000,ip actions=drop
 cookie=0x0, duration=323.864s, table=0, n_packets=12, n_bytes=936, idle_age=112, priority=0 actions=NORMAL
/ #
```

Figura 33 – Lista de regras das inserções via FWunify no OpenFlow

```
/ #
/ # ovs-ofctl dump-flows br0
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=385.120s, table=0, n_packets=0, n_bytes=0, idle_age=385, priority=65535,ip,nw_dst=200.19.0.20 actions=dr
op
 cookie=0x0, duration=384.067s, table=0, n_packets=0, n_bytes=0, idle_age=384, priority=65534,tcp,nw_src=10.0.0.0/24,tp_dst=44
3 actions=NORMAL
 cookie=0x0, duration=383.060s, table=0, n_packets=0, n_bytes=0, idle_age=383, priority=65533,tcp,nw_src=10.0.0.0/24,tp_dst=80
 actions=NORMAL
 cookie=0x0, duration=382.054s, table=0, n_packets=0, n_bytes=0, idle_age=382, priority=65532,tcp,nw_src=10.0.0.10,nw_dst=200.
19.0.10,tp_dst=3306 actions=NORMAL
 cookie=0x0, duration=381.047s, table=0, n_packets=0, n_bytes=0, idle_age=381, priority=65531,tcp,nw_src=10.0.0.0/24,nw_dst=20
0.19.0.10,tp_dst=3306 actions=drop
 cookie=0x0, duration=380.042s, table=0, n_packets=0, n_bytes=0, idle_age=380, priority=65530,ip actions=drop
 cookie=0x0, duration=379.036s, table=0, n_packets=0, n_bytes=0, idle_age=379, priority=0 actions=NORMAL
/ #
```

APÊNDICE H – TRADUÇÃO E APLICAÇÃO DAS 38 ACLS

A Figura 34 apresenta 8 exemplos políticas ACL na forma de intenções FWlang. Estas intenções fazem parte do conjunto de 38 regras[1] ativas no firewall de borda (Cisco) de uma das instituições utilizadas nas nossas análises. As 8 intenções servem como exemplo da utilização do FWunify para aplicação de regras reais. A Figura 35 apresenta a relação de regras do firewall da instituição (os IPs originais formam preservados por questões de segurança) e a Figura 36 apresenta a relação de regras geradas pela FWunify. Como podemos observar, as relações de regras inseridas manualmente (pelos administradores do firewall da instituição) e automaticamente (via FWunify) são idênticas.

Figura 34 – Exemplos de intenções FWlang

```
define intent acl:
  name  text(drop-all-all)
  from  endpoint(any)
  to    endpoint(any)
  deny  traffic(any)
  order after(all-intents)
  add   firewall(cisco-1)
```

(a) drop-all-all

```
define intent acl:
  name  text(permit-all-logserver)
  from  endpoint(any)
  to    endpoint(200.19.0.155)
  allow traffic(syslog)
  order before(all-intents)
  add   firewall(cisco-1)
```

(b) permit-all-logserver

```
define intent acl:
  name  text(deny-all-smtp)
  from  endpoint(any)
  to    endpoint(any)
  deny  traffic(smtp)
  order before(all-intents)
  add   firewall(cisco-1)
```

(c) deny-all-smtp

```
define intent acl:
  name  text(permit-net60-all)
  from  range(10.0.60.0/24)
  to    endpoint(any)
  allow traffic(any)
  order before(all-intents)
  add   firewall(cisco-1)
```

(d) permit-net60-all

```
define intent acl:
  name  text(permit-all-ftp)
  from  endpoint(any)
  to    endpoint(any)
  allow traffic(ftp)
  order before(all-intents)
  add   firewall(cisco-1)
```

(e) permit-all-ftp

```
define intent acl:
  name  text(permit-all-https)
  from  endpoint(any)
  to    endpoint(any)
  allow traffic(https)
  order before(all-intents)
  add   firewall(cisco-1)
```

(f) permit-all-https

```
define intent acl:
  name  text(deny-net90-all)
  from  range(10.0.90.0/24)
  to    endpoint(any)
  deny  traffic(any)
  order before(all-intents)
  add   firewall(cisco-1)
```

(g) deny-net90-all

```
define intent acl:
  name  text(deny-incident01)
  from  endpoint(any)
  to    endpoint(85.243.251.19)
  deny  traffic(any)
  order before(all-intents)
  add   firewall(cisco-1)
```

(h) deny-incident01

[1] O conjunto completo representado com FWlang está disponível em <https://github.com/mmfiorenza/fwunify>

Figura 35 – Lista das 38 regras da instituição

```
fw-campus02# show access-list
access-list cached ACL log flows: total 0, denied 0 (deny-flow-max 4096)
            alert-interval 300
access-list inside_access_in; 38 elements; name hash: 0x433a1af1
access-list inside_access_in line 1 extended deny ip any host 85.243.251.19 (hitcnt=0) 0x0d40fc5f
access-list inside_access_in line 2 extended deny ip any host 106.184.21.218 (hitcnt=0) 0xee33fd24
access-list inside_access_in line 3 extended deny ip any host 37.140.235.126 (hitcnt=0) 0xdb98d420
access-list inside_access_in line 4 extended deny ip any host 37.140.235.132 (hitcnt=0) 0x10aa95f5
access-list inside_access_in line 5 extended permit ip host 10.0.20.4 10.100.0.0 255.255.255.0 (hitcnt=0) 0xf4758959
access-list inside_access_in line 6 extended deny ip any 10.100.0.0 255.255.255.0 (hitcnt=0) 0xa804fe71
access-list inside_access_in line 7 extended permit tcp host 10.0.90.10 host 200.19.0.12 eq ldap (hitcnt=0) 0x5aec0f0a
access-list inside_access_in line 8 extended permit tcp host 10.0.90.10 host 10.100.0.103 eq 50000 (hitcnt=0) 0x8b77ecb7
access-list inside_access_in line 9 extended permit tcp host 10.0.90.10 any eq 5938 (hitcnt=0) 0xef55c7ce
access-list inside_access_in line 10 extended deny ip 10.0.90.0 255.255.255.0 any (hitcnt=0) 0x9b530f98
access-list inside_access_in line 11 extended permit tcp any any eq https (hitcnt=0) 0x6908ba56
access-list inside_access_in line 12 extended permit tcp any any eq www (hitcnt=0) 0x386bad81
access-list inside_access_in line 13 extended permit tcp any any eq 8080 (hitcnt=0) 0x448c3b8c
access-list inside_access_in line 14 extended permit tcp any any eq 8090 (hitcnt=0) 0xc6abe459
access-list inside_access_in line 15 extended permit udp any any eq 8090 (hitcnt=0) 0x08ab07bf
access-list inside_access_in line 16 extended permit tcp any any eq 6500 (hitcnt=0) 0x7afe4d0b
access-list inside_access_in line 17 extended permit tcp any any eq 8443 (hitcnt=0) 0xac6cf297
access-list inside_access_in line 18 extended permit tcp any any eq 7443 (hitcnt=0) 0x0baf2d04
access-list inside_access_in line 19 extended permit tcp any any eq 9191 (hitcnt=0) 0x65641e04
access-list inside_access_in line 20 extended permit tcp any any eq 3001 (hitcnt=0) 0x50b1e4b6
access-list inside_access_in line 21 extended permit tcp any any eq ftp (hitcnt=0) 0xd04b582a
access-list inside_access_in line 22 extended permit tcp 10.0.20.0 255.255.255.0 host 10.100.0.50 eq 3389 (hitcnt=0) 0x64f8a61c
access-list inside_access_in line 23 extended permit tcp 10.0.30.0 255.255.255.0 host 10.100.0.50 eq 3389 (hitcnt=0) 0x5eb42185
access-list inside_access_in line 24 extended permit tcp 10.0.40.0 255.255.255.0 host 10.100.0.50 eq 3389 (hitcnt=0) 0x5b0c6ced
access-list inside_access_in line 25 extended permit ip 10.0.60.0 255.255.255.0 any (hitcnt=0) 0x2cdb4a71
access-list inside_access_in line 26 extended permit tcp host 10.0.10.254 host 200.19.0.50 eq 1812 (hitcnt=0) 0x8b9c0745
access-list inside_access_in line 27 extended permit udp host 10.0.10.254 host 200.19.0.50 eq 1812 (hitcnt=0) 0xa0580e9b
access-list inside_access_in line 28 extended deny tcp any any eq smtp (hitcnt=0) 0xe3de3aa9
access-list inside_access_in line 29 extended permit udp 10.0.0.0 255.255.0.0 host 200.19.0.111 eq sip (hitcnt=0) 0xacb6c9eb
access-list inside_access_in line 30 extended permit tcp any any eq 1935 (hitcnt=0) 0x7ee61e81
access-list inside_access_in line 31 extended permit udp 10.0.0.0 255.255.0.0 host 10.100.0.33 eq 4040 (hitcnt=0) 0xe9eaad0e
access-list inside_access_in line 32 extended permit tcp 10.0.0.0 255.255.0.0 host 10.100.0.33 eq 4040 (hitcnt=0) 0x4b39f930
access-list inside_access_in line 33 extended permit tcp any host 200.19.0.155 eq 3128 (hitcnt=0) 0x661bd265
access-list inside_access_in line 34 extended permit tcp 10.0.0.0 255.255.0.0 any eq ssh (hitcnt=0) 0x45cea8f0
access-list inside_access_in line 35 extended permit udp any host 200.19.0.155 eq syslog (hitcnt=0) 0xda007d58
access-list inside_access_in line 36 extended permit tcp 10.0.0.0 255.255.0.0 any eq 1688 (hitcnt=0) 0x594889c4
access-list inside_access_in line 37 extended permit tcp any any eq 3456 (hitcnt=0) 0xd9f60634
access-list inside_access_in line 38 extended deny ip any any (hitcnt=0) 0xbe9efe96
fw-campus02#
```

Figura 36 – Lista das 38 regras adicionadas pela FWunify

```
firewall-1-cisco# show access-list
access-list cached ACL log flows: total 0, denied 0 (deny-flow-max 4096)
            alert-interval 300
access-list inside_access_in; 38 elements; name hash: 0x433a1af1
access-list inside_access_in line 1 extended deny ip any host 85.243.251.19 (hitcnt=0) 0x0d40fc5f
access-list inside_access_in line 2 extended deny ip any host 106.184.21.218 (hitcnt=0) 0xee33fd24
access-list inside_access_in line 3 extended deny ip any host 37.140.235.126 (hitcnt=0) 0xdb98d420
access-list inside_access_in line 4 extended deny ip any host 37.140.235.132 (hitcnt=0) 0x10aa95f5
access-list inside_access_in line 5 extended permit ip host 10.0.20.4 10.100.0.0 255.255.0.0 (hitcnt=0) 0x18f611f0
access-list inside_access_in line 6 extended deny ip any 10.100.0.0 255.255.0.0 (hitcnt=0) 0x116e8253
access-list inside_access_in line 7 extended permit tcp host 10.0.90.10 host 200.19.0.12 eq ldap (hitcnt=0) 0x5aec0f0a
access-list inside_access_in line 8 extended permit tcp host 10.0.90.10 host 10.100.0.103 eq 5000 (hitcnt=0) 0x1575ef77
access-list inside_access_in line 9 extended permit tcp host 10.0.90.10 any eq 5938 (hitcnt=0) 0xef55c7ce
access-list inside_access_in line 10 extended deny ip 10.0.90.0 255.255.255.0 any (hitcnt=0) 0x9b530f98
access-list inside_access_in line 11 extended permit tcp any any eq https (hitcnt=0) 0x6908ba56
access-list inside_access_in line 12 extended permit tcp any any eq www (hitcnt=0) 0x386bad81
access-list inside_access_in line 13 extended permit tcp any any eq 8080 (hitcnt=0) 0x448c3b8c
access-list inside_access_in line 14 extended permit tcp any any eq 8090 (hitcnt=0) 0xc6abe459
access-list inside_access_in line 15 extended permit udp any any eq 8090 (hitcnt=0) 0x08ab07bf
access-list inside_access_in line 16 extended permit tcp any any eq 6500 (hitcnt=0) 0x7afe4d0b
access-list inside_access_in line 17 extended permit tcp any any eq 8443 (hitcnt=0) 0xac6cf297
access-list inside_access_in line 18 extended permit tcp any any eq 7443 (hitcnt=0) 0x0baf2d04
access-list inside_access_in line 19 extended permit tcp any any eq 9191 (hitcnt=0) 0x65641e04
access-list inside_access_in line 20 extended permit tcp any any eq 3001 (hitcnt=0) 0x50b1e4b6
access-list inside_access_in line 21 extended permit tcp any any eq ftp (hitcnt=0) 0xd04b582a
access-list inside_access_in line 22 extended permit tcp 10.0.20.0 255.255.255.0 host 10.100.0.50 eq 3389 (hitcnt=0) 0x64f8a61c
access-list inside_access_in line 23 extended permit tcp 10.0.30.0 255.255.255.0 host 10.100.0.50 eq 3389 (hitcnt=0) 0x5eb42185
access-list inside_access_in line 24 extended permit tcp 10.0.40.0 255.255.255.0 host 10.100.0.50 eq 3389 (hitcnt=0) 0x5b0c6ced
access-list inside_access_in line 25 extended permit ip 10.0.60.0 255.255.255.0 any (hitcnt=0) 0x2cdb4a71
access-list inside_access_in line 26 extended permit tcp host 10.0.10.254 host 200.19.0.50 eq 1812 (hitcnt=0) 0x8b9c0745
access-list inside_access_in line 27 extended permit udp host 10.0.10.254 host 200.19.0.50 eq 1812 (hitcnt=0) 0xa0580e9b
access-list inside_access_in line 28 extended deny tcp any any eq smtp (hitcnt=0) 0xe3de3aa9
access-list inside_access_in line 29 extended permit udp 10.0.0.0 255.255.0.0 host 200.19.0.111 eq sip (hitcnt=0) 0xacb6c9eb
access-list inside_access_in line 30 extended permit tcp any any eq 1935 (hitcnt=0) 0x7ee61e81
access-list inside_access_in line 31 extended permit udp 10.0.0.0 255.255.0.0 host 10.100.0.33 eq 4040 (hitcnt=0) 0xe9eaad0e
access-list inside_access_in line 32 extended permit tcp 10.0.0.0 255.255.0.0 host 10.100.0.33 eq 4040 (hitcnt=0) 0x4b39f930
access-list inside_access_in line 33 extended permit tcp any host 200.19.0.155 eq 3128 (hitcnt=0) 0x661bd265
access-list inside_access_in line 34 extended permit tcp 10.0.0.0 255.255.0.0 any eq ssh (hitcnt=0) 0x45cea8f0
access-list inside_access_in line 35 extended permit udp any host 200.19.0.155 eq syslog (hitcnt=0) 0xda007d58
access-list inside_access_in line 36 extended permit tcp 10.0.0.0 255.255.0.0 any eq 1688 (hitcnt=0) 0x594889c4
access-list inside_access_in line 37 extended permit tcp any any eq 3456 (hitcnt=0) 0xd9f60634
access-list inside_access_in line 38 extended deny ip any any (hitcnt=0) 0xbe9efe96
firewall-1-cisco#
```