UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Diego Couto de Carvalho

# DETECÇÃO DE INTRUSÕES EM DISPOSITIVOS DE REDE COM O FILTRO DE PACOTE BERKELEY

Santa Maria, RS
2023

**Diego Couto de Carvalho**

# DETECÇÃO DE INTRUSÕES EM DISPOSITIVOS DE REDE COM O FILTRO DE PACOTE BERKELEY

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação (PGCC), da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Mestre em Ciência da Computação.**

Orientador: Prof. Dr. Carlos Raniery P. Dos Santos

Santa Maria, RS

2023

**Diego Couto de Carvalho**

# DETECÇÃO DE INTRUSÕES EM DISPOSITIVOS DE REDE COM O FILTRO DE PACOTE BERKELEY

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação (PGCC), da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Mestre em Ciência da Computação.**

**Aprovado em 04 de 08 de 2023:**

---

**Carlos Raniery P. Dos Santos, Dr.**
(Presidente/Orientador)

---

**Raul Ceretta Nunes, Dr. (UFSM)**

---

**Luis Alvaro De Lima Silva, Dr. (UFSM)**

Santa Maria, RS

2023

# AGRADECIMENTOS

Agradeço aos meus pais e avós paternos que me apoiaram e incentivaram para que o caminho da computação fosse trilhado, permitindo o início da paixão pela computação até a conclusão da graduação na área.

A minha esposa e filha que auxiliaram no decorrer deste trabalho estando ao meu lado, me incentivando e ajudando de inúmeras maneiras desde o início.

Ao meu orientador Prof. Dr. Carlos Raniery Paula dos Santos pela sua disponibilidade nas reuniões, orientações, sugestões, discussões e revisões acerca da elaboração deste trabalho.

Aos amigos e colegas Marcelo da Luz Colomé e Augusto Gouvêa Weber que disponibilizaram um tempo para discutir alguns pontos importantes presentes na elaboração deste documento. E também aos colegas servidores da divisão de pós graduação Guilherme Alves Elwanger e Marcos Alexandro Vargas Mello, pela amizade construída, bom humor e por auxiliarem nos processos exigidos para realização desse trabalho de mestrado.

A UFSM por ser esta universidade de excelência da qual faço parte como servidor, que me possibilitou crescer, estudar e me qualificar.

*“Modern digital applications and services are increasingly designed and implemented according to microservice patterns”*

(JOANNA KOŁODZIEJ, 2022, P. 12 )

# RESUMO

## DETECÇÃO DE INTRUSÕES EM DISPOSITIVOS DE REDE COM O FILTRO DE PACOTE BERKELEY

AUTOR:
DIEGO COUTO DE CARVALHO
ORIENTADOR: CARLOS RANIERY P. DOS SANTOS

Sistemas de detecção de intrusões modernos são comumente desenvolvidos com uso de algoritmos de aprendizado de máquina e seleção de atributos. No entanto, o custo computacional desses algoritmos limita a capacidade de resposta imediata às intrusões. Isso acontece porque esses algoritmos são geralmente executados em controladores que são dispositivos centralizados, os quais processam as informações recebidas de sondas na rede, visando obter um resultado de classificação para o tráfego de rede analisado. Como resultado, podem surgir problemas como atraso na identificação de ameaças, além da possibilidade de sobrecarga nesse dispositivo centralizado caso os ataques estejam ocorrendo em diversos dispositivos simultaneamente. Neste trabalho, é proposta uma arquitetura para detecção de intrusões em tempo real em dispositivos de rede compatíveis com eBPF a partir de modelos otimizados assincronamente através de uma estratégia de seleção de atributos com vistas a otimizar modelos de ML. Tal otimização é necessária para lidar com as restrições impostas pela tecnologia eBPF quando executada em dispositivos de rede, as quais impõem limitações nos programas gerados quanto à memória, funções e ao tamanho do programa a ser executado. Essa arquitetura visa configurar um classificador que possa ser empregado em dispositivos de rede como switches. Dessa maneira, a classificação pode ser realizada diretamente nesses equipamentos, eliminando a necessidade das sondas que enviam informações para os controladores. Como prova de conceito, um modelo pode ser construído, partindo de um computador que realiza a avaliação e configuração de um classificador compatível com dispositivos eBPF no Kernel do sistema Linux. Os resultados obtidos revelam que a solução proposta é capaz de detectar e prevenir intrusões em tempo real com baixa sobrecarga para os cenários avaliados.

**Palavras-chave:** Incidentes de Segurança. Detecções de intrusões. Segurança da Informação. Sistemas de Resposta à Intrusão. Inteligência Artificial.

# ABSTRACT


## INTRUSION DETECTION ON NETWORK DEVICES WITH THE BERKELEY PACKET FILTER

AUTHOR:
DIEGO COUTO DE CARVALHO
ADVISOR: CARLOS RANIERY P. DOS SANTOS

Modern intrusion detection systems are commonly developed using machine learning algorithms and feature selection. However, the computational cost of these algorithms limits the ability to respond immediately to intrusions. This is because these algorithms are typically run on controllers, which are centralized devices that process information received from probes on the network in order to obtain a classification result for the analyzed network traffic. As a result, problems such as delays in threat identification may arise, as well as the possibility of overloading the centralized device if attacks are occurring on multiple devices simultaneously. In this work, an architecture for real-time intrusion detection on network devices compatible with eBPF is proposed, using models optimized asynchronously through a feature selection strategy to optimize ML models. Such optimization is necessary to deal with the constraints imposed by eBPF technology when executed on network devices, which impose limitations on generated programs in terms of memory, functions, and program size to be executed. This architecture aims to configure a classifier that can be employed on network devices such as switches. In this way, classification can be performed directly on these devices, eliminating the need for probes that send information to controllers. As a proof of concept, a model can be constructed, starting from a computer that performs the evaluation and configuration of a classifier compatible with eBPF devices in the Linux system kernel. The results obtained reveal that the proposed solution is capable of detecting and preventing intrusions in real-time with low overhead for the evaluated scenarios.



**Keywords:** Security incidents. Intrusion detections. Information security. Intrusion Response Systems. Artificial intelligence.

# LISTA DE FIGURAS

# LISTA DE TABELAS

# LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| IDS | Sistema de Detecção de Intrusões ou Intrusion Detection System |
| DoS | Denial of Service |
| DDoS | Distributed Denial of Service |
| VM | Virtual Machine |
| NFV | Network Function Virtualization |
| SDN | Software Defined Network |
| NIDS | Network Intrusion Detection System |
| HIDS | Host Intrusion Detection System |
| MAC | Media Access Control |
| BNN | Binary Neural Network |
| RF | Randon Forest |
| KNN | K Nearest Neighbors |
| ML | Machine Learning |
| ASIC | Application Specific Integrated Circuit |
| FPGA | Field Programmable Gate Array |
| P4 | Programming Protocol-Independent Packet Processors |
| BPF | Berkeley Packet Filter |
| eBPF | extended Berkeley Packet Filter |
| bmv2 | Behavioral Model Second Version |
| P4C | P4 Compiler |
| IP | Internet Protocol |
| UDP | User Datagram Protocol |
| ICMP | Internet Control Message Protocol |
| TCP | Transfer Control Protocol |
| SYN | Synchronize |
| ACK | Acknowledgement |

# SUMÁRIO

# 1 INTRODUÇÃO

A expansão das redes de computadores e também dos incidentes de segurança têm despertado a atenção da indústria e do meio acadêmico na busca por sistemas de segurança eficientes. Apesar das organizações e empresas utilizarem diferentes tipos de sistemas de segurança como firewalls, antivírus, criptografia e autenticação de usuários, muitas são vítimas de ataques cibernéticos (ZHOU et al., 2020). Uma das formas de garantir mais segurança é o uso de um Sistema de Detecção de Intrusões (IDS), um mecanismo de segurança para detecção de atividades ilegais em computadores e sistemas de rede, sendo vital para garantir a segurança da rede e seus ativos associados, uma vez que monitora e detecta invasores que buscam comprometer a sua integridade, confidencialidade e disponibilidade (KHRAISAT et al., 2019).

Os IDSs podem ser categorizados de acordo com a abordagem de detecção em duas categorias: detecção de anomalia e detecção de mau-uso/assinaturas. Os sistemas de detecção de assinaturas verificam as atividades atuais analisadas com assinaturas armazenadas advindas de ataques conhecidos, de maneira que é utilizado o conhecimento de ataques anteriores na formação de regras usadas na sua detecção. Já no que se refere a detecção de anomalia, pode definir-se como uma abordagem de detecção de intrusões que primeiramente aprende a característica de uma atividade normal para detectar quaisquer características que desviem desse comportamento, pois, nesse tipo de abordagem, dados de comportamentos normais são usados na construção de modelos utilizados na detecção de atividades diferentes das quais estão nos dados observados. Considerando aumentar a segurança, ambas as abordagens de detecção de intrusões são utilizadas na identificação de ataques (DEPREN et al., 2005).

Além da abordagem de detecção, IDSs também podem ser distinguidos baseados na fonte de informação que é analisada por estes sistemas, ou seja, um sistema de detecção pode coletar informações que trafegam na rede sendo conhecido como *Network Intrusion Detection System* (NIDS) ou então coletam informações de *hosts* específicos sendo conhecido como *Hosts Intrusion Detection System* (HIDS). Dessa maneira, HIDS entrega uma visão do que acontece especificamente em um determinado *host*. Enquanto NIDS, permite uma analise geral de todos os dispositivos conectados a localidades da rede através de sensores que são posicionados para monitorar o tráfego. A vantagem de IDS baseados em rede (NIDS) é que um único sensor pode monitorar diversos dispositivos de rede facilitando a detecção e implantação desse sistema. Entretanto, esses sensores requerem uma comunicação com um controlador central

podendo apresentar problemas de processamento e desempenho da rede, principalmente em redes grandes ou sobrecarregadas, devido à necessidade de troca de informações entre sensores e controlador (FERNANDES et al., 2019).

Por outro lado, em geral NIDSs necessitam de equipamentos que processem pacotes com alto desempenho. Portanto, empresas desenvolvem equipamentos middlebox de sua propriedade individual. Esses equipamentos são especializados na detecção de anomalias e permitem um alto desempenho do funcionamento da rede, entretanto, o custo deles é alto, além de apresentarem flexibilidade limitada em aspectos como funcionalidades, capacidade e localização dos dispositivos. Tais limitações resultam em custos econômicos significativos com a necessidade de atualização de hardware caso os equipamentos não consigam detectar novos vetores de ataques (LI et al., 2021).

A busca por uma solução que alcance flexibilidade e redução de custos operacionais, levou ao surgimento de novas tecnologias como Redes Definidas por Software - SDN e Virtualização de funções de Rede - NFV. Tais soluções permitem flexibilidade através da programabilidade baseada em software, sendo possível usar o NFV para atribuir a máquinas virtuais (VMs) um sistema de defesa baseado em características de ataques específicos e aproveitar o SDN para encaminhar o tráfego às VMs que forem mais adequadas para lidar com determinadas características de ataques (FAYAZ et al., 2015). Embora, essas tecnologias permitam personalizar o processamento dos pacotes conforme a especificação do administrador da rede, elas são utilizadas em processadores de uso geral e portanto elas possuem desvantagem devido à capacidade de processamento ser inferior a hardwares especializados que podem sustentar Terabyte de tráfego de rede. Dessa forma, em segurança de rede é necessário que a detecção seja rápida, porém esse requisito fica comprometido uma vez que o tráfego precisa ser encaminhado a um controlador central que realizará o processamento dos dados e a identificação de comportamentos suspeito. Ademais, um fabricante pode oferecer equipamentos dedicados que podem não atender satisfatoriamente a ataques diferentes, enquanto, NFV poderia ser flexível para trocar soluções que atenderiam a ataques variados, mas sem o mesmo desempenho (ZHANG et al., 2020).

O paradigma SDN reduziu a complexidade da rede permitindo o desenvolvimento de soluções através de software para o plano de controle da rede. Entretanto, o plano de dados ainda permanece com funções fixas e restrito as especificações do protocolo OpenFlow (MCKEOWN et al., 2008), que necessita de novas atualizações para adicionar novos recursos (*i.e.*,

parsers diferentes do especificado na versão), permanecendo então o plano de dados limitado as especificações da versão do protocolo. Em busca da solução dessa limitação do plano de dados surgiu a linguagem P4 (Programming Protocol-Independent Packet Processors), que permitiu aos usuários programarem switches, tal como era possível apenas aos desenvolvedores dos equipamentos, resultando em aplicações variadas (*i.e*., firewalls, balanceadores de carga) (ALSABEH et al., 2022). Com o surgimento do P4, vários fabricantes de chips adotaram a tecnologia, possibilitando que os usuário possam experimentar novas ferramentas e protocolos customizados, incluindo grandes empresas como Facebook e Google que otimizam suas redes usando plataformas e aplicações baseadas em P4 (KFOURY; CRICHIGNO; BOU-HARB, 2021).

Outra tecnologia utilizada no processamento de pacotes é o *extended Berlekely Packet Filter* (eBPF), uma máquina virtual executada no Kernel, amplamente utilizada na filtragem de pacotes e monitoramento de sistemas. Com o uso do eBPF é possível processar pacotes nas camadas mais baixas da pilha de rede usando o *hook Express Data Path* (XDP). O eBPF/XDP apresenta diversos benefícios como processamento rápido, baixa sobrecarga de processamento (WANG; CHANG, 2022a). Ademais, o eBPF é uma linguagem de programação de baixo nível (similar ao código de máquina), vindo a ser empregada em diversas aplicações do processamento de pacotes de redes com algumas vantagens em relação a outras linguagens como P4, entre elas está a possibilidade de reconfigurar completamente um programa em tempo de execução, sem precisar reiniciar ou interromper o serviço e também a comunicação bilateral entre espaço do usuário (pode se ter aplicações de controle) e Kernel (processamento no plano de dados) (TU; RUFFY; BUDIU, 2018).

Desse modo, a programação no Kernel do sistema mostra-se como um opção viável no desenvolvimento de uma solução de segurança, uma vez que se tem obtido um melhor desempenho quando comparada a soluções que são executadas no espaço do usuário (PRADHAN; MANNEPALLI, 2021), (WANG; CHANG, 2022a). Nesse contexto, o *extended Berkeley Packet Filter* (eBPF) permite implementar programas que podem ser executados diretamente no Kernel do Linux e executados diretamente em Placas de redes inteligentes (SmartNICs), habilitando a programação no plano de dados para esses dispositivos e a flexibilidade na implementação de funções de rede (VIEIRA et al., 2019).

Para resolver o problema de segurança em um ambiente onde o desempenho é um fator crítico para o processamento de pacotes, faz-se necessário utilizar equipamentos com altas taxas

de transferência no processamento de pacotes (WANG; CHANG, 2022a). O XDP vem sendo adotado em ambientes de rede em produção, pois fornece programabilidade com alto desempenho para o processamento de pacotes em hardwares comerciais (DIMOLIANIS; PAVLIDIS; MAGLARIS, 2021). Ademais, o plano de dados programável possibilita ao usuário customizar o funcionamento da rede, escrevendo seus próprios programas, os quais podem ser implantados ou removidos do dispositivo de rede sem a necessidade de atualizar hardware ou depender do fabricante do equipamento (HAUSER et al., 2021). Portanto, alguns autores implementam funções de redes no plano de dados programável utilizando eBPF/XDP, sendo que através dos programas executados no Kernel do sistemas é possível aumentar a taxa de transferência de dados em relação a programas executados no espaço do usuário. Técnicas de aprendizado de máquina são amplamente utilizadas na solução de assertividade em diferentes abordagens de IDSs, entretanto, devido a limitações de hardware e software, esses algoritmos dificilmente seriam portabilizados para um programa eBPF. Portanto muitos autores utilizam técnicas baseadas em assinaturas que são limitadas a reconhecer ataques conhecidos, outros autores empregam Machine Learning (ML), porém, não apresentam resultados de detecção. Um outro ponto que pode ser considerado é que, devido ao eBPF ser injetado em um dispositivo de rede, pode fornecer a função de IDS em dispositivos existentes na rede não necessitando de hardware adicional.

Nesse contexto, o presente trabalho propõe uma arquitetura para traduzir bases de dados em modelos de classificação com ML, utilizados para configurar um Sistema de Detecção de Intrusões no plano de dados. As principais contribuições desse trabalho são: a implementação de uma adaptação do algoritmo KNN compatível para dispositivos no plano de dados programável, utilizando a tecnologia eBPF e a geração do código-fonte do programa referente ao classificador produzido. Nessa arquitetura, a construção e otimização (*e.g.,* seleção de atributos, normalização dos dados) do modelo acontece exclusivamente em um computador externo e sem limitações (Espaço do Usuário). O trabalho também apresenta uma análise detalhada sobre a aplicação de IDS no contexto de Redes Definidas por Software (SDN), incluindo uma variedade de estudos relacionados ao tema. O classificador, por sua vez, além de ser utilizado na fase de treinamento, também é executado no Kernel do sistema Linux através do eBPF, portanto, é compatível com interfaces de rede ou switches com suporte a eBPF. A arquitetura proposta foi desenvolvida em ambiente simulado, gerando resultados capazes de revelar que a solução proposta é passível de detectar ataques em tempo real e não exige muitos recursos computacionais, portanto apresenta uma baixa sobrecarga em diferentes cenários avaliados.

Assim, este documento está organizado da seguinte maneira; no Capítulo 2 é apresentada a fundamentação teórica, no Capítulo 3 são discutidos e mostrados os trabalhos relacionados. Já o Capítulo 4 apresenta os detalhes da implementação da arquitetura proposta neste trabalho, enquanto, no Capítulo 5 são mostrados os detalhes da implementação da arquitetura. Em seguida, no Capítulo 6 são descritos os experimentos realizados, além dos seus resultados. Por fim, a conclusão é exposta no Capítulo 7.

# 2 FUNDAMENTAÇÃO TEÓRICA

O atual Capítulo tem por finalidade discutir os principais conhecimentos abordados deste trabalho, as principais tecnologias adotadas no processamento de pacotes no plano de dados e algumas formas de uso dessas tecnologias para garantir segurança em um ambiente de rede onde elas são utilizadas.

## 2.1 PROGRAMABILIDADE NO PLANO DE DADOS

Pode-se caracterizar redes de computadores como uma infraestrutura de importância crítica em empresas privadas ou instituições governamentais. Essa infraestrutura é projetada para funcionar em diversos ambientes e atender a uma variedade de casos de uso. Enquanto se busca otimizar dispositivos de rede para funções específicas, também é importante considerar a redução de custos. A necessidade em desenvolver dispositivos de rede de uso genérico, com baixo custo de produção, ganhou extrema importância a ponto de pesquisadores se interessarem e testarem novas ideias para essa infraestrutura de rede. Entretanto, tradicionalmente, os equipamentos e os protocolos eram fixos não permitindo que novas ideias fossem testadas, levando a comunidade acadêmica a buscar soluções inovadoras no desenvolvimento de redes programáveis. Devido à essa demanda por sistemas mais flexíveis para atender ao uso generalizado da rede, tornou-se necessário o desenvolvimento de arquiteturas de rede e ativos de rede (*i.e.*, dispositivos de rede) mais flexíveis. Em contrapartida ao corriqueiro funcionamento da indústria em que os equipamentos eram fabricados com funções de redes fixas e permitiam pouca ou nenhuma programabilidade, seria caro e demorado para o fabricante atender a novas necessidades de protocolos e aplicações (MICHEL et al., 2021).

Tal limitação nos dispositivos de encaminhamento no plano de dados ocorre devido à maneira pela qual tradicionalmente são desenvolvidas as funções, os protocolos e a programabilidade nos equipamentos de rede tradicionais. Nesse tipo de equipamento, o usuário fica limitado a lógica de processamento definida no dispositivo, ou seja, o funcionamento e opções são definidos pelo fabricante. Então, a ação de adicionar novas funcionalidades em equipamentos tradicionais não pode ser realizada pelo usuário, ficando a cargo do fabricante desenvolver funções, testar e implantar componentes em hardware dedicados, podendo levar de 2 ou 3 anos para que esse ciclo de desenvolvimento seja concluído. Em consequência da demora e in-

flexibilidade do ciclo de desenvolvimento, novas funções são implementadas apenas quando necessárias a uma quantidade massiva de usuários. Ademais, a inovação fica limitada devido à implementação estar limitada aos fabricantes e o desenvolvimento das funções que são demandadas em larga escala (GAO; WANG, 2021). Também pode ser considerado que mesmo que o fabricante pudesse desenvolver todos os protocolos, funções de rede e inseri-los na lógica de processamento de um dispositivo, isso não seria viável devido ao custo de memória, ciclos de processamento e funcionalidades que não seriam utilizadas. Em passos da solução desse impasse, uma nova geração de dispositivos de redes foi introduzida, permitindo a sua reconfiguração através de técnicas de programabilidade (BIFULCO; RÉTVÁRI, 2018).

Uma alternativa possível pode ser alcançada com o protocolo Openflow, que, em sua concepção, foi desenvolvido para permitir a separação física do plano de controle e do plano de encaminhamento, fazendo com que equipamentos (*i.e.*, switches e roteadores) executassem o processamento de pacotes da maneira que foi definida pelo plano de controle. Tal abordagem permite que o plano de controle seja removido desses dispositivos para serem disponibilizados em ponto(s) centralizado(s), sendo responsável por definir a maneira como os pacotes são processados pelo plano de dados. O Openflow foi pensado como um protocolo aberto para programar as tabelas de encaminhamentos de diferentes switches e roteadores. Dessa forma, esse protocolo possui um controle que é definido pela lógica de encaminhamento, que pode ser definida externamente pelo controlador Openflow através de um canal de comunicação seguro (MCKEOWN et al., 2008). Ademais, o protocolo Openflow permitiu a flexibilização do plano de controle, uma vez que esse plano pode ser implementado em componentes externos com recursos abundantes. Entretanto, o plano de dados mantém-se engessado, devido à necessidade do protocolo, no qual o fabricante define quais cabeçalhos e ações são suportadas, não permitindo protocolos personalizados, além da necessidade de que novas extensões do protocolo sejam implementadas caso exista a demanda por protocolos mais recentes (GUTIÉRREZ et al., 2021). A falta da possibilidade de customização, por exemplo, não permite a definição de contadores personalizados, que permitiriam uma coleção e agregação de metadados que poderiam ser úteis em aplicações diversas (GRAY et al., 2021).

Portanto, o desenvolvimento do OpenFlow como uma implementação da abordagem Redes Definidas por Softwares (SDN) tradicional foi um fato de grande importância. Nesse tipo de rede, são implementados componentes centrais (controlador) no monitoramento, coleta de dados e também na realização de detecções. Os dados podem ser coletados diretamente dos dis-

positivos de encaminhamento, entretanto, as operações de rede estão restritas às configurações de protocolos e cabeçalhos suportados por esses hardwares. Ademais, ainda existe um problema com o uso dessa tecnologia devido à alta demanda de comunicação entre o plano de dados e o plano de controle. Essa comunicação entre os dois planos poderá causar problemas tais como gargalos na comunicação, demora na identificação de ameaças e dificuldade de processamento devido ao plano de controle não conseguir inspecionar pacotes em uma velocidade semelhante ao plano de dados, que tem capacidade de inspecionar com maior velocidade (GRAY et al., 2021).

No que se refere ao processamento no plano de dados, é possível atribuir destaque aos equipamentos dedicados conhecidos como *Middlebox*, que são hardwares projetados para implementar serviços como firewalls, balanceadores de carga ou mesmo hardware de segurança com recursos de IDS diretamente no plano de dados, que é o foco desse trabalho. Esses *Middlebox* aumentam os custos de operação e também dos ativos da rede, uma vez que ficam atrelados a fabricantes lançarem atualização, esses custos podem ser elencados caso a demanda de tráfego de rede seja alta. Considerando os aspectos mencionados, uma outra solução para diminuir o alto custo de operação fornecido por *Middlebox* e obter flexibilidade para criar aplicações, pode ser alcançada através de tecnologia de Virtualização de Funções de Rede (NFV). No entanto, essa solução implementada em processadores de uso geral não é suficiente para alcançar um alto desempenho no processamento de pacotes, devido à necessidade de mecanismos mais adequados do que simplesmente separar as tarefas em threads de execução simultânea (CEROVIĆ et al., 2018). Além disso, essa abordagem também pode causar atrasos na rede, uma vez que esses equipamentos realizam o processamento de pacotes nas camadas altas de aplicações do sistema operacional (VIEIRA et al., 2019). Na busca por uma solução que proporcionasse um menor custo, alto desempenho no processamento de pacotes e ainda flexibilização na implementação serviços de rede no plano de dados, foram introduzidas tecnologias como eBPF e P4, possibilitando o desenvolvimento de ferramentas através de linguagens de programação, que podem ser utilizadas nos próprios equipamentos que processam pacotes (i,e,. switches, routers, dispositivos de rede), obtendo o desempenho necessário para a alta demanda de processamento de pacotes no plano de dados (GRAY et al., 2021).

Nesse contexto, a programabilidade no plano de dados possibilita que a lógica de processamentos de pacotes em baixo nível de um dispositivo de rede seja sistematicamente, rapidamente e compreensivamente reconfigurada de maneira padronizada. A programabilidade pode

ser vista também como funções fixas convencionais, nas quais o usuário pode modificar o processamento de pacotes. Portanto, dispositivos programáveis permitem que o usuário modifique o encaminhamento de pacotes conforme a sua necessidade (MICHEL et al., 2021). Ademais, a programabilidade também pode ser definida como a possibilidade de um usuário escrever algoritmos que podem ser executados por hardwares e softwares. Em adicional, a programabilidade no contexto de redes de computadores atribuem ao plano de dados, por exemplo, a capacidade de executar algoritmos na rede e modificar a lógica de processamento de pacotes, modificando a função do plano de dados de maneira simplificada, de tal forma que usuários e fabricantes possam escrever algoritmos e alterar o funcionamento da rede conforme sua necessidade e especificações de desempenho (GAO; WANG, 2021).

Recentemente a programabilidade era realizada em tempo de compilação, isto é, os novos programas e modificações no plano de dados eram aplicados no dispositivo antes dele processar o tráfego com as novas configurações, ocorrendo uma interrupção de serviço ao aplicar as novas regras, por isso, introduziram-se equipamentos que possibilitam a programabilidade em tempo de execução. Considerando-se o que foi dito anteriormente, são destacadas, nesse texto três categorias de dispositivos programáveis. A primeira e mais limitada das categorias apresentadas é a dos switches *Application-Specific Integrated Circuit* (ASICs) reconfiguráveis , que possibilitam a programabilidade em tempo de execução através de modificações nas tabelas match/action. A segunda modalidade de dispositivos são as *Field-Programmable Gate Arrays* (FPGAs) e SmartNICs baseadas em *System-on-a-Chip* (SoC-based), sendo esta mais flexível que os switches ASICs, pois possibilita a reconfiguração em tempo de compilação e execução. Nesse tipo de dispositivo, a programabilidade em tempo de execução fica a cargo do programa que o dispositivo carrega. Por exemplo, quando executa um programa P4 é possível reconfigurar tabelas e parsers. A última categoria apresentada é a dos hosts baseados em Kernel, sendo essa a mais flexível das apresentadas, pois através do Kernel é possível escrever programas em linguagem C restrita e injetar esses programas na pilha de rede através da extensão de Kernel eBPF. Ademais, o eBPF pode atender a necessidade de programabilidade em tempo de execução (Reconfigurações e recarregar diferentes programas), ou seja, permite a substituição de programas na pilha de rede sem qualquer interrupção no serviço (XING et al., 2021) (MIANO et al., 2018).

A

## 2.2 EBPF

Dados os meios apresentados, pode-se afirmar que programas executados em Kernel permitem uma maior flexibilidade para programabilidade, uma vez que possibilitam adicionar ou remover programas e ainda alterar configurações dos programas executados em tempo de execução, ou seja, sem a necessidade de reiniciar o dispositivo ou interromper o serviço. Assim sendo, *Berkeley Packet Filter*, referenciado como BPF clássico (cBPF), é amplamente conhecido por ser uma linguagem de filtro de pacotes utilizada pelo tcpdump. O cBPF foi proposto por Steven McCanne and Van Jacobson, em 1992, como uma linguagem de programação que possibilita que um conjunto de instruções seja executado em uma máquina virtual no Kernel do sistema Linux. Inicialmente o cBPF era muito útil para filtragem de pacotes, porém, a comunidade logo percebeu o potencial do cBPF para outras áreas e portanto melhorias foram realizadas tanto na máquina virtual quanto na sua arquitetura geral. Essas melhorias geraram uma nova versão chamada de extended BPF (eBPF), que foi introduzida a partir da versão 3.15 do Kernel do Linux. Entre as melhorias implementadas está o aumento do número de registradores de 2 para 11, mapas eBPF (tabelas de tamanho fixo, contendo chaves e valores), funções auxiliares e adicionou o recurso tail-calls, que permite contornar a limitação de tamanho de um programa eBPF, através da execução de um outro programa eBPF. Também foram adicionados conjuntos de instruções lógicas e aritméticas, comportando classes de 32 e 64 Bytes para as ULA (Unidade Lógica Aritmética). Ambas as classes da ULA suportam operações matemáticas de soma, subtração, E lógico, OU lógico, deslocamentos à esquerda, deslocamentos à direita, OU exclusivo, multiplicação, divisão, resto e negação, além da função de chamada de função do kernel (VIEIRA et al., 2019). Após a introdução do eBPF, ele foi rapidamente adotado em ferramentas como monitoramento de rede e sistemas, manipulação do tráfego de rede, balanceamento de carga, etc. O crescimento do uso do eBPF foi tanto que até mesmo empresas como Facebook, Netrone, Cilium o utilizam em projetos.

O eBPF é uma eficiente máquina virtual que executa bytecodes escritos pelo usuário em linguagem de alto nível, sendo compilados em linguagem de baixo nível pelo compilador eBPF. O bytecode executado pelo eBPF pode ser utilizado em pontos de *hooks* do Kernel do sistema, de maneira que consegue interceptar informações do sistema ou pacotes de rede de computadores, garantindo que esse programa seja executado de maneira segura (KEHOE, 2022). Através do eBPF, o desenvolvedor pode escrever uma aplicação na linguagem C ou P4 e depois compilá-

la para instruções eBPF. O código eBPF resultante pode ser processado no kernel do Linux ou por dispositivos programáveis como NetFPGAs e placas de interface de rede inteligente (SmartNICs) (VIEIRA et al., 2019).

Tal prática é realizada através de um código de Bytes (bytecode), transferido do espaço do usuário ao Kernel do sistema. Durante essa abordagem são realizadas etapas de verificações, prevenindo que problemas de segurança e falhas no kernel ocorram, portanto, embora o desenvolvedor consiga escrever programas para propósitos gerais utilizando o conjunto de instruções eBPF, é preciso garantir que esse programa não danifique o funcionamento do Kernel. Após as verificações, o programa será anexado a um soquete e irá processar pacote a pacote garantido que o seu processamento possa ser modificado com flexibilidade e segurança (VIEIRA et al., 2019).

Um exemplo de bytecode eBPF pode ser visualizado na Figura 1, para um melhor entendimento do que é feito nesse código eBPF é possível abrir a ferramenta Wireshark, selecionando um pacote com protocolo TCP que possibilita visualizar os Bytes do pacote (Packet Bytes) conforme a Figura 2.

Figura 1 – Exemplo de bytecode de um programa eBPF.

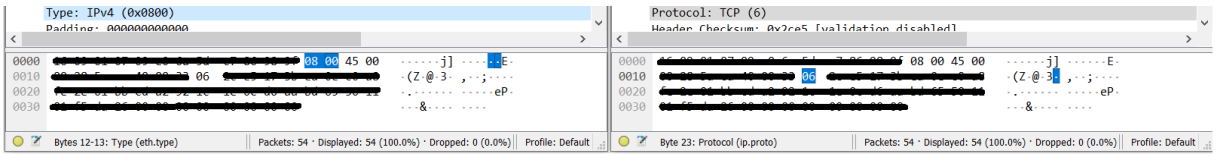
```
000 - load 2 bytes @ [12]
001 - jump equal #0x800 jump true 2 jump false 5
002 - load 1 byte @ [23]
003 - jump equal #6 jump true 4 jump false 5
004 - return #-1
005 - return #0
```

Fonte: Adaptado de (VIEIRA et al., 2019).

A Figura 1 apresenta um exemplo de programa eBPF, que realiza as seguintes instruções; Na linha 000 é carregado o offset 12 do quadro, gravando a informação eth.type no acumulador (Figura 2 esquerda). Em 001 o valor que foi carregado no acumulador é comparado com o valor 0x800 e, caso verdadeiro, a execução do programa pula para a instrução 002, caso contrário, o salto é para a instrução 005, encerrando sua execução com retorno do valor 0. A instrução 002 carrega o offset 23 do quadro, gravando a informação ip.proto no acumulador ( Figura 2 direita). Em seguida, na instrução 003, o valor gravado no acumulador será comparado ao valor 6, que é referente ao protocolo IPV4, caso a comparação seja verdadeira o programa salta para a instrução 005, sendo encerrado com retorno 0, caso contrário, o salto será para a instrução 004 retornando -1 (VIEIRA et al., 2020).

Após a compilação do programa e posterior inserção no dispositivo, o programa eBPF é

Figura 2 – Visualização dos Bytes de um pacote da ferramenta Wireshark, nessa imagem o quadro a direita mostra o offset 12, enquanto o quadro a direita mostra o offset 23.




Fonte: Software Wireshark.

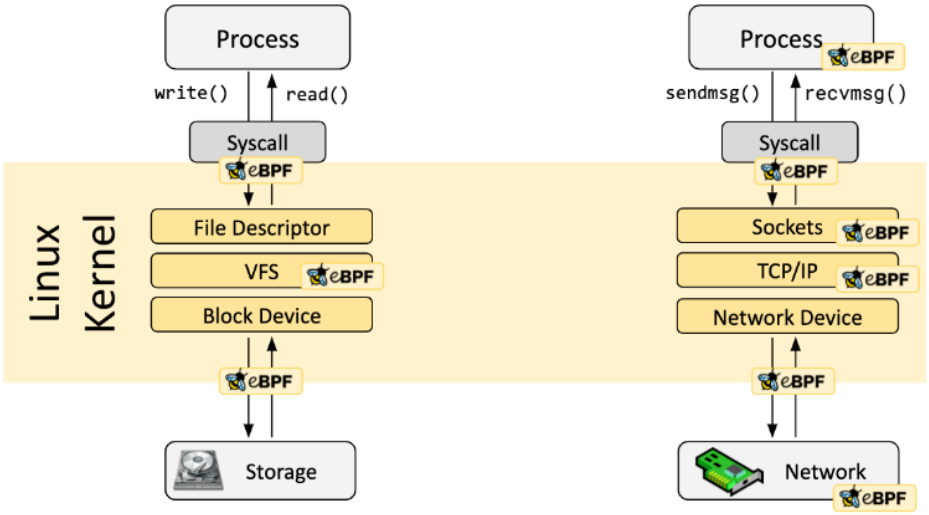
executado em determinados eventos do sistema, que são acionados quando um dado de entrada ou saída atravessa um *hook point*. *Hooks points* incluem chamadas do sistema, funções de entrada/saída, pontos de rastreamento do kernel, eventos de rede entre outros. A Figura 3, mostra alguns dos possíveis *hook points* que podem ser utilizados para interceptar eventos acionando um programa eBPF (EBPF, 2023). O *hook* também pode ser apresentado como uma interface que permite ao programador customizar a programação do Kernel (VIEIRA et al., 2019), uma dessas interfaces é o *eXpress Data Path* (XDP), responsável por permitir que desenvolvedores instalem programas que processam pacotes no kernel do Linux (MIANO et al., 2018).

Basicamente, o Linux possibilita que programas eBPF possam ser inseridos no kernel, coletem informações de dados e manipulem eventos de maneira customizada. No contexto das redes de computadores, *hooks* são usados para interceptar os pacotes antes ou durante as chamadas do sistema operacional. Existem outros *hooks* eBPF como o *Linux Security Module* (LSM), utilizado para fins de segurança de hosts na auditoria de eventos de segurança (KEHOE, 2022) e também o *Traffic Control* (TC) que assim como XDP é um *hook* de interceptação de pacotes, entretanto o XDP possibilita que esses pacotes sejam interceptados diretamente no driver da interface de rede, enquanto o *hook* TC processa os pacotes posteriormente a alocação do *Linux Socket Buffer* (skb) (MIANO et al., 2018).

Embora um programa eBPF seja limitado, não contando com todas as bibliotecas disponíveis na linguagem C, por exemplo, ele conta com alguns *Helpers* que podem ser utilizados para realizar a interação entre os *hooks*, funções do kernel e estruturas, tais como tabelas de roteamento, mecânicos de túneis, mapas eBPF etc. Mapas eBPF, por sua vez, são um local de armazenamento genérico disponível em programas eBPF, pois podem ser usados no armazenamento de informações coletadas durante o processamento dos pacotes. As informações guardadas nos mapas podem ser acessadas através de uma chave, observando que tanto a chave quando a informação acessível pela chave são números binários, além disso, também existe a obrigação de informar o tamanho do mapa durante sua definição na escrita do código fonte.

A criação de um mapa eBPF pode ocorrer através da definição da estrutura **struct**

Figura 3 – Visualização dos points Hooks eBPF.




Fonte: Adaptado de (FOUNDATION, 2023).

**bpf_map_def**, na qual se deve declarar o tipo de mapa utilizado, o tamanho da chave em bits, o valor contido na chave e o número máximo de entradas. Um exemplo de definição pode ser observado na estrutura declarada na Figura 4. Nessa definição é passado o tipo de mapa eBPF, o tamanho da chave de acesso em Bytes, o número de Bytes que irá armazenar as informações e o número de linhas ou de chaves que o mapa comportará. Assim, também é informado o parâmetro *sizeof* da estrutura presente na Figura 5, que, conforme pode ser observado, tem alguns atributos com tamanho de Bytes de maneira que o tamanho total dessa estrutura será alocada em *value_size* na definição do mapa. Outro parâmetro definido na estrutura é o *type* de mapa, um tipo de array. Outros mapas com comportamentos e estruturas diferentes podem ser declarados, alguns são genéricos e outros mais específicos, conforme amostra listada a seguir (MAURIZIO, 2017):

- BPF_MAP_TYPE_HASH.
- BPF_MAP_TYPE_PERCPU_HASH.
- BPF_MAP_TYPE_PERCPU_ARRAY.
- BPF_MAP_TYPE_PROG_ARRAY.
- BPF_MAP_TYPE_PERF_EVENT_ARRAY.
- BPF_MAP_TYPE_STACK_TRACE.

Figura 4 – Definição do mapa eBPF.

```
struct bpf_map_def SEC("maps") nslkdd_mapa_selecionado = {
        .type        = BPF_MAP_TYPE_ARRAY,
        .key_size    = sizeof(__u32),
        .value_size  = sizeof(struct knn_features_selecionadas),
        .max_entries = 7,
};
```

Fonte: Autoria própria.

Figura 5 – Struct seleção duas features KNN.

```
    struct knn_features_selecionadas {
        __u64 protocol_type ;
        __u64 service ;
        __u8 classe ;

    };
```

Fonte: Autoria própria.

Conforme enunciado nessa sessão, o eBPF é um programa executado de maneira segura no kernel do Linux. Tal segurança é fornecida através do verificador que é utilizado pelo Kernel na verificação das intruções eBPF que serão carregadas para o sistema, garantindo integridade e segurança ao sistema operacional. A aferição é realizada pelo verificador disponível no código fonte do kernel (kernel/bpf/verifier.c), que é responsável por analisar se o programa não é maior do que o permitido, se ele possui um fim, se a memória alocada pelo programa está dentro do espaço permitido, se os *loops* possuem tamanhos definidos, entre outras verificações. Ao fim desse processo, o verificador decidirá se o programa eBPF será aceito ou não.

Destarte, o eBPF é uma abordagem que pode ser utilizada como alternativa para flexibilizar o plano de dados programável devido ao alto desempenho no processamento de pacotes e também ser usado em hardwares comerciais (DIMOLIANIS; PAVLIDIS; MAGLARIS, 2021), tendo em vista que um programa será injetado em uma máquina virtual restrita no Kernel do Linux, a qual deverá garantir a integridade do sistema. Esse programa pode ser codificado diretamente usando instruções assemble ou então convertidos para bytecode usando a ferramenta bpm_asm ou, ainda, linguagem C restrita compilada através do compilador LLVM Clang. Desse modo, o eBPF é afetado por limitações amplamente conhecidas (*i.e* Tamanho do programa 4096 Bytes, laços de repetição limitados). Ainda assim, é vantajoso devido a possibilidade de o eBPF/XDP proporcionar a reconfiguração em tempo de execução, além de seu funcionamento

ocorrer nas camadas mais baixas da pilha de rede, evitando processamento em camadas mais altas, o que seria custoso devido à alocação de maiores recursos computacionais. Portanto o eBPF obtém significativos benefícios de desempenho em prol de limitações (MIANO et al., 2018). Ademais, o eBPF possui a vantagem de possibilitar a comunicação bidirecional entre o espaço do usuário e o kernel, uma vez que ambos podem efetuar leituras e escritas em um mapa eBPF. Isso difere da linguagem P4, na qual o plano de dados não pode gravar, ficando restrito apenas à leitura de informações dos registradores e contadores (TU; RUFFY; BUDIU, 2018).

## 2.3 IA EM SEGURANÇA DE REDES

No tocante a redes de comunicação, a segurança da rede pode ser provida de diferentes maneiras, dentre elas o uso de firewall, um software de segurança que impede o acesso não autorizados de serviços do sistema, criptografia e restrição de acesso ao usuário. Ainda assim, a rede pode ficar suscetível a ameaças se as credenciais do usuário puderem ser acessadas. Uma medida de segurança adicional pode ser feita através de IDS (AMROLLAHI et al., 2020). Esse sistema é um software ou hardware que monitora o tráfego de rede verificando comportamentos suspeitos, violações de políticas e ocorrências de ameaças conhecidas na intenção de enviar alarmes caso detecte algum problema. Desse modo, é possível definir que as duas abordagens mais utilizadas na detecção de ataques são baseadas em mau-uso e anomalias. Nesse caso, a abordagem de mau-uso reconhece padrões registrados de atividades maliciosas e a de anomalia detecta o desvio do bom comportamento) (UMAR; ZHANFANG, 2020).

Somando-se a essa discussão, em geral, a classificação de anomalias pode ser alcançada com algoritmos de aprendizado de máquina por agrupamento de amostras de tráfego ou dados de aplicações similares (QUINCOZES et al., 2020). Para que esse IDS consiga identificar anomalias, são realizadas etapas de treinamento e teste. Durante a etapa de treinamento, dados são usados para que o sistema aprenda um modelo de comportamento normal. Já na etapa de teste, novos dados são usados para estabelecer a capacidade do sistema em prever novas ameaças (KHRAISAT et al., 2019).

Os algoritmos árvores de decisão (RF), K-vizinhos mais próximos (KNN) e Rede Neural (RNN) podem ser utilizados na verificação do comportamento de sistemas computacionais, destacando-se por ter uma boa assertividade no contexto redes de computadores (VINAYAKUMAR et al., 2019). Entretanto, devido às limitações impostas a linguagens de programação nos dispositivos de encaminhamento, não seria trivial implementar uma RNN. Nesses aspectos as

opções de RF e KNN podem ser implementadas no plano de dados.

De fato, é sabido que métodos baseados em ML fornecem alta acurácia e são massivamente utilizados para classificações de pacotes. Porém, o processamento desses algoritmos é realizado em um *host* ou servidor remoto (QIN et al., 2020a). Portanto, um dos problemas amplamente discutido nos trabalhos relacionados à segurança de redes de computadores é o local onde os dados serão processados para alimentar IDS. Dessa maneira, IDS são amplamente discutidos no contexto de redes programáveis, sofrendo algumas críticas por parte dos pesquisadores, pois a maioria das soluções desenvolvidas para detecção de intrusos são executadas no plano de controle. Essas soluções geralmente possuem duas desvantagens, sendo a primeira devido às configurações de atributos como contadores padrões ou eventos de pacotes, que são insuficientes para resultados razoáveis de acurácia em algoritmos de ML; a segunda é que os algoritmos são tipicamente uma tarefa que requer bastante capacidade computacional em operações matemáticas e seleção de atributos, podendo causar atrasos e interromper o correto funcionamento da rede (GUTIÉRREZ et al., 2021).

Por esse motivo, algumas estratégias aproveitam a flexibilidade do plano de dados programável para melhorar a acurácia e o tempo de processamento dos algoritmos de ML (GUTIÉRREZ et al., 2021). Dentre os métodos utilizados, há aqueles nos quais os pesquisadores tentam diminuir o atraso de processamento no plano de controle através da implementação de algumas funções diretamente no plano de dados, por exemplo, há abordagens que delegam ao plano de dados a responsabilidade de entregar atributos ao IDS, com a aplicação dessa estratégia é possível diminuir o atraso do IDS no plano de controle uma vez que o plano de dados consegue repassar os atributos especificados com maior agilidade. Outra forma de tentar aumentar o desempenho de sistemas IDS pode ocorrer pela configuração de regras no plano de dados, já que podem ser usadas na filtragem do tráfego de rede, descartando tráfego malicioso ou ainda com a finalidade de escolher quais pacotes são processados pelo controlador (GUTIÉRREZ et al., 2021). Além dessas opções é possível, também, diminuir o atraso no processamento de IDS implementando o algoritmo diretamente no plano de dados, conforme o realizados pelos autores (PRADHAN; MANNEPALLI, 2021), (BACHL; FABINI; ZSEBY, 2021), (BEN-YAIR; ROGOVOY; ZAIDENBERG, 2019), (WANG; CHANG, 2022b) e (CARVALHO BERTOLI et al., 2020) que implementam essa função com resultados promissores utilizando programas no Kernel do sistema (i,e,. eBPF). Embora ainda esteja em escopo limitado devido às restrições impostas pelos equipamentos que trabalham no encaminhamento dos pacotes. Outro ponto a ser

salientado é que, devido ao eBPF não possuir bibliotecas de ML próprias, é necessário implementar todas as fases do algoritmo para realizar comparações entre Kernel e espaço do usuário.

Tal como é visto nas tecnologias abordadas e também devido às restrições impostas ao programa eBPF no Kernel do sistema Linux, a parte pesada do processamento é feita no espaço do usuário. A maioria dos trabalhos em eBPF para segurança de redes de computadores utiliza assinaturas na implementação de um sistema IDS e, portanto, esses trabalhos não conseguem identificar ataques caso eles não estejam em sua base de assinaturas. O ML que pode representar o comportamento do usuário e, por isso, identificar novos ataques, desponta em fase inicial com alguns estudos (PRADHAN; MANNEPALLI, 2021) e (BACHL; FABINI; ZSEBY, 2021). De outro modo, o ML é usado para realizar a seleção de assinaturas e implantar representações simplificadas delas nos dispositivos de Kernel (*i.e.*, switches, roteadores, hosts).

# 3 TRABALHOS RELACIONADOS

Neste Capítulo são discutidos os trabalhos relacionados. Primeiramente, o estado da arte de sistemas de detecção de intrusos é abordado, segregando em quais pontos da rede eles funcionam e quais características são usadas na identificação do tráfego de rede. Logo após, são apresentadas abordagens utilizadas em sistemas IDS incluindo a identificação de ataques diretamente no plano de dados ou realizado por dispositivo externo, destacando o seu funcionamento, vantagens e desvantagens.

## 3.1 ANÁLISE DE ABORDAGENS EM SISTEMAS DE DETECÇÃO

Entre as técnicas comumente utilizadas no desenvolvimento de IDS, pode-se destacar a localização do detector de intrusos além do modo como a detecção é realizada (Abordagem de detecção/Fonte de Informação). Nesses termos, entre as diferente ópticas utilizadas na literatura podemos encontrar dois tipos de abordagens que são utilizadas pelos autores. Primeiramente, pode-se destacar a técnica mais comum utilizada, onde detectores são elencados externamente ao plano de dados para seleção de regras e filtros no plano de dados. Uma segunda metodologia usadas pelos autores, são aquelas em que o plano de dados processa informações como estatísticas, entropias ou atributos com a possibilidade da detecção ocorrer diretamente no plano de dados ou então esses atributos serem enviados até um processamento externo para uma avaliação mais robusta.

### 3.1.1 Detecção realizada fora do plano de dados - Plano de dados executando Filtros e Assinaturas

Nesta seção são abordados trabalhos que realizam algum processamento externo, ou seja, no plano de controle, no propósito de definir regras ou filtros empregados no plano de dados. Efetuou-se a revisão de abordagens nas quais são realizados procedimentos externos na definição de um conjunto de regras e filtros, que são utilizadas no plano de dados em diferentes propostas de autores. Embora o plano de dados suporte processar regras e identificar ataques conhecidos por meio delas, a identificação dessas regras é realizada em procedimentos externos.

(ZHANG et al., 2020) propõe uma abordagem nomeada como POSEIDON a fim de detectar vários tipos de ataques volumétricos DDoS. Em POSEIDON são definidas políticas

que podem ser implementadas pelo operador da rede através de uma linguagem simples, capaz de detectar alguns ataques no plano de dados. Na intenção de tornar esse trabalho adaptativo, o autor propõe uma extensão em (LI et al., 2021). Nesse novo trabalho, os estados de conexões são enviados periodicamente a um controlador externo que monitora essas conexões. O monitoramento é realizado através de um gráfico que possui políticas implementadas com definições de ataques (*i.e.*, estatísticas), usadas para selecionar as regras de detecção do plano de dados. De maneira semelhante, (KUKA et al., 2019) propõe uma abordagem para mitigação do ataque de amplificação, baseado em volumes de IPS (*i.e.*, IP de origem). Nessa abordagem, o plano de dados é responsável por coletar as estatísticas e as enviar ao plano de controle, que realiza uma verificação a fim de detectar ataques. Caso ocorra alguma detecção, o plano de controle define instruções ao plano de dados bloqueando o tráfego através das regras recebidas.

Uma outra abordagem com processamento externo ocorre em (BERTIN, 2017), que desenvolve o sistema de mitigação Gatebot. Nesse sistema, um componente central recebe amostras de tráfego e as analisa através de um mecanismo baseado em limiares na identificação do tráfego que será mitigado. Na ocasião em que ocorre a detecção de algum ataque, uma regra de mitigação será inserida no Kernel eBPF, que consiste em um componente localizado na borda da rede para mitigar ataques. Nesses trabalhos, todo o processo de detecção é baseado em regras ou assinaturas, definidas externamente de maneira automatizada através de algum mecanismo externo de identificação dos ataques, com exceção de (ZHANG et al., 2020) no qual as regras pertencentes a ataques são descritas e escolhidas manualmente. Esses trabalhos possuem a capacidade de prever ataques conhecidos pelas assinaturas contidas em seu sistema, mesmo com as técnicas que automatizam a seleção das assinaturas. Entretanto, em novos ataques, é preciso especificar as assinaturas assim como o procedimento de seleção das mesmas.

O trabalho de (LEWIS; BROADBENT; RACE, 2019) é um esquema composto de regras baseadas no IDS Snort para evitar o processamento de parte dos pacotes por um IDS externo. P4ID é composto pelos componentes Rule Parse e implementação P4. O componente Rule parser traduz regras do snort em match-action p4 que são instaladas com um programa P4 no plano de dados, através dessa abordagem são encaminhados para o IDS externo apenas pacotes que coincidem com as assinaturas. Outro trabalho baseado em assinatura (CARVALHO BERTOLI et al., 2020) apresenta uma abordagem para prevenção de ataques do tipo probe. Essas assinaturas são geradas externamente para ser implementadas em filtros eBPF/XDP e NetFilter ao nível de Kernel do sistema. Em ambos os trabalhos os pacotes são filtrados no plano de dados

conforme as regras instaladas, enquanto a seleção dessas regras é feita externamente.

É proposto por (DIMOLIANIS; PAVLIDIS; MAGLARIS, 2021) um componente XDP no Kernel do sistema que extrai atributos importantes selecionados previamente. Eles são espelhados até um outro componente no espaço do usuário que usa ML para identificar assinaturas maliciosas. Caso ocorra a detecção, as assinaturas classificadas são enviadas a outro componente (espaço do usuário) redutor de assinaturas, que as otimiza combinando características importantes de um conjunto de assinaturas selecionadas para enviar as que foram reduzidas ao componente de mitigação de anomalias executada no Kernel do sistema. O autor (WANG; CHANG, 2022a) recorre a uma análise prévia no plano de dados na identificação de pacotes suspeitos. No trabalho é implementado um programa executado no kernel do sistema, analisando se os payloads dos pacotes contêm padrões similares às regras simplificadas do IDS/IPS Snort. Caso o pacote não coincida com essas regras ele volta ao fluxo de rede normal. Se coincidir, o pacote será analisado por um outro programa eBPF executado no espaço do usuário, que decidirá qual ação será realizada com esse pacote. Através dessa abordagem, o autor consegue obter um desempenho três vezes superior ao do Snort. Ambas abordagens aproveitam o processamento rápido do plano de dados na melhoria do desempenho de seus sistemas, entretanto, a dependência da aplicação de análise externa pode ocasionar problemas relacionados a troca de informações entre o plano de dados e a aplicação central.

Alguns autores utilizam ML realizando o processamento externo em sua abordagem de detecção aproveitando o plano de dados para alimentar algum módulo de detecção. Nesse aspecto, (GRAY et al., 2021) apresenta uma abordagem inspirada no trabalho de (LEWIS; BROADBENT; RACE, 2019), tentando ampliar os ataques identificados pelo detector. Nesse trabalho a extração de metadados e coleta de atributos é realizada pelo plano de dados em um programa P4 executado em um switch. Os dados extraídos do switch são coletados pela aplicação *controller* que os entrega à aplicação *receiver* responsável pelo armazenamento e processamento dos atributos, além de gerar um arquivo CSV. Por fim, a aplicação *detection* que é um detector munido com o algoritmo *Random Forest* treinado com um dataset público, recebe o arquivo CSV contendo fluxos completos de rede usados na identificação de ataques. O autor (MUSUMECI et al., 2020) realiza no plano de dados a coleta de informações em janelas de tempo as enviando a um detector externo. Tal detector utiliza ML na detecção de ataques do tipo TCP flood, alimentado com informações recebidas de switches na rede. Além da apresentação da solução, são avaliados os algoritmos ML (RF, KNN e SVN), que são processados no

detector externo ao plano de dados. O trabalho é estendido pelo autor em (MUSUMECI et al., 2022), nessa nova abordagem são apresentadas as arquiteturas Standalone DAD (*Detection Attack* DDoS) e Correlated DAD, de modo que na primeira arquitetura é colocado um módulo de detecção em cada switch e na segunda arquitetura existe apenas um módulo de detecção que recebe informações dos switches. Além disso o autor apresenta novas métricas como F1-Score e acurácia. Entretanto o trabalho é limitado a detectar ataques DDoS que são possíveis de identificar através da contagem de pacotes TCP, UDP e *Synchronize* (SYN) TCP em uma janela de tempo formando os atributos analisados. Com uma abordagem inspirada nesses dois trabalhos (ROSHANI; NOBAKHT, 2022) apresenta o HybridDAD como um framework que é capaz de detecar uma maior variedade de ataques DDoS volumétricos, através da análise de outros atributos como a contagem de pacotes *Acknowledgement* (ACK) do tipo UDP. Esses trabalhos dependem da detecção realizada externamente, o que pode ocasionar gargalos e atrasos na rede. A maioria deles é capaz de detectar apenas ataques DDoS, entretanto (GRAY et al., 2021) pode identificar uma maior variedade de ataques.

### 3.1.2 Detecção diretamente no plano de dados

Nesta subseção são apresentados trabalhos que realizam a detecção diretamente no plano de dados, mediante a apresentação de técnicas que exploram o uso de estatísticas, entropias, taxas ou ainda o uso de tecnologias mais aprimoradas como o ML na detecção de anomalias na rede ou ainda o ajuste de parâmetros.

Estatísticas podem ser usadas na identificação de ataques por meio da verificação do desvio de comportamento padrão. Nesses termos, (SANTOYO-GONZÁLEZ; CERVELLÓ-PASTOR; PEZAROS, 2019) apresenta uma abordagem de segurança no gateway da rede (plausível em switches, roteadores). A solução efetua a coleta de estatísticas (*i.e.*, quantidade de pacotes, taxas, volumes e distribuição de tamanho) as utilizando em um cálculo de entropia e pesos exponenciais baseados em limiares para a detecção de ataques DDoS. Outra abordagem é proposta por (GAO; HANDLEY; VISSICCHIO, 2021). Os autores implementam uma biblioteca para analisar estatísticas (*i.e.*, Volumes de pacotes SYNs por IP) capazes de detectar ataques DDoS volumétricos. O uso da biblioteca é factível em switches programáveis, proporcionando que esses equipamentos processem algumas estatísticas (*i.e.*, média, variância, desvio padrão, percentual de distribuição) de tráfego de rede.

Outros autores utilizam Sketch em abordagens na identificação de ameaças à rede. Dessa

maneira, o trabalho de (TAVARES; FERRETO, 2019) apresenta uma solução baseada em Sketch para prevenir e detectar ataques DDoS volumétricos. A abordagem é implementada em P4, utilizando o algoritmo SACK2 apresentado em (SUN et al., 2009), capaz de detectar ataques do tipo TCP SYN Flood. Nessa proposta, um limiar é verificado e caso ele seja alcançado, novas tentativas de conexão serão realizadas por intermédio de uma validação de hash, consequentemente os atacantes serão bloqueados. (KHOOI et al., 2020) apresenta *Distributed In-network Defense Architecture* (DIDA), uma solução de detecção e mitigação para ataques DDoS. DIDA é uma arquitetura distribuída baseada em Sketchs, que funciona no plano de dados monitorando conexões através de contadores, além de manter uma estrutura de dados Sketch com dados correlacionados de requisições e respostas dessas conexões, sendo usada na identificação de ataques. Quando um ataque é identificado, o IP do atacante é adicionado a uma Lista de Acesso de Controle (ACL) gerenciada pelo plano de controle, que também a compartilha com os outros dispositivos gerenciados. (DING et al., 2021) apresenta uma estratégia para identificação de vítimas de ataques DDoS volumétricos. Esse trabalho utiliza modelos probabilísticos baseados na quantidade de contatos realizados para um host em determinado intervalo de tempo. A estratégia combina bitmaps (ESTAN; VARGHESE; FISK, 2003) e Count-min Sketch (CMS) (CORMODE; MUTHUKRISHNAN, 2005) para formar um novo Sketch (uma estrutura de dados probabilística) nomeado como BACON Sketch, capacitado para estimar a quantidade de fluxos distintos contactam um determinado host. Ademais, é apresentada uma estratégia chamada de INDDOS que aproveita o BACON Sketch na identificação da ocorrência de uma alta taxa de requisições de IPs de origem até um determinado IP destino de um determinado host por determinado período de tempo. Esses trabalhos são capazes de identificar uma alta variação de requisições, entretanto, podem apresentar problemas se um determinado serviço aumentar a quantidade de solicitações repentinamente ou em ataques não volumétricos.

Entropias também são utilizadas na identificação dos riscos que prejudicam a correta operação da rede, nesses termos, em (LAPOLLI; MARQUES; GASPARY, 2019) é utilizado um cálculo de entropia diretamente no plano de dados para verificar o desvio de comportamento padrão em ataques DDoS. A entropia é calculada através do monitoramento dos IPs de origem e destino, no qual são gerados cálculos de entropias baseados em frequências das requisições desses IPs. Uma vez gerada a entropia que caracteriza o tráfego normal da rede, são gerados alarmes no caso de desvio desse comportamento. Também baseada na observação de requisições de IPs, porém com participação do operador na decisão das ações resultantes de de-

tecções, (SILVEIRA ILHA et al., 2020) apresenta um sistema de defesa para ataques baseados em anomalias, no qual são usadas medidas de entropia de Shannon na identificação do tráfego anômalo. Na presente abordagem, um limite é definido com finalidade de observar a contagem de requisições dos IPs de origem e destino. O sistema retem dois estados de funcionamento que ativam e desativam uma defesa, permitindo ao operador definir se o tráfego anômalo será descartado, suprimido ou distraído. Também baseado na observação de requisições, (DING; SAVI; SIRACUSA, 2021) apresenta o framework P4DDoS. Nesse trabalho são consideradas as limitações do plano de dados na definição de um modo de detecção de ataques. Portanto, o tráfego suspeito é identificado através de um cálculo de entropia baseado em IP destino e origem dos pacotes recebidos. O autor (GOLCHIN et al., 2022) propõe uma abordagem de detecção de ataques DDoS SYN flood no plano de dados P4, baseado em entropia que é calculada com base no número de sequências de *Handshakes* não finalizadas. O autor também menciona que em redes congestionadas é comum que o *Handshake* seja prejudicado, com isso algumas requisições normais são descartadas erroneamente pelo sistema, entretanto, a abordagem aproveita o envio de pacotes SYNs duplicados em determinados períodos de tempo para mitigar o problema. Esses trabalhos, em sua maioria, utilizam o volume de requisições na detecção de ataques. Ademais, o autor (SILVEIRA ILHA et al., 2020) permite ao operador uma tomada de decisão, enquanto (GOLCHIN et al., 2022) se diferencia dos demais destacando conexões não finalizadas e também visualiza problemas de retransmissão em redes congestionadas. Embora os trabalhos tenham uma rápida resposta a incidentes, ainda são restritos a ataques volumétricos não prevendo outros ataques ou uma alta demanda repentina de usuários.

Atributos baseados em probabilidade e contadores são usados por (DIMOLIANIS; PAVLIDIS; MAGLARIS, 2020), na proposta de uma abordagem que opera no plano de dados para detectar ataques DDoS. Nesse trabalho, durante uma janela de tempo nomeado como época, são contabilizados três atributos baseados em: (i) porcentagens de fluxo do tráfego de entrada das *subnets* monitoradas, (ii) fluxos recebido por *subnets*, (iii) relação proporcional entre fluxos de entradas e saída. A detecção é baseada na quantidade de tentativas de conexões realizadas a uma determinada subrede monitorada, com a vantagens de relacionar a proporção entre entradas e saídas na tentativa de evitar que alguma *subnet* recebendo uma alta quantidade de tráfego válido realize uma detecção de falso positivo, a qual acionaria sistemas externos de mitigação. Em uma abordagem diferente, na qual métrica de taxas por pacotes são utilizadas na detecção e mitigação de ataques DoS *spoofing* (falsificação) e DoS volumétrico, é apresentada por

(SIMSEK et al., 2020). Baseada na observação de ataques direcionados ao controlador, que ocasionam o aumento da comunicação dele com os switches, saturando suas tabelas de encaminhamentos. Nesses termos, o sistema de defesa calcula um valor de hash utilizando endereço MAC e IP de origem dos pacotes na identificação do host conectado a porta, comparando se o pacote subsequente tem o mesmo hash. Além disso, é usado o *Two Rate Three Color Marker* (RFC 2698) um esquema que se baseia em duas taxas de contagem de pacotes para definir se apenas envia um alerta ou bloqueia e envia o alerta ao controlador. Nesse estudo existe uma proteção para IPs falsificados, além da escalada na ação de mitigação realizada, embora seja limitada a ataques DoS.

Algumas abordagens realizam algum tipo de detecção no plano de dados, entretanto os parâmetros de detecção são configurados externamente. Nesse aspecto (BARRADAS et al., 2021) apresenta o sistema FlowLens, que se baseia no *Flow Marks Accumalor* (FMA) na identificação de ameaças. Nesse sistema, o FMA é uma representação compacta dos dados baseada no tamanho dos pacotes e a distribuição de tempo entre recebimento de pacotes nos quais *flow markes* acumulam fluxos de rede. Ademais, o operador da rede pode definir a característica do funcionamento de aplicações específicas, tais quais a detecção de conversas botnet, *Website fingerprinting* (reconhecimento de sites criptografados através de coleta das distribuições do comprimento dos fluxos de pacotes), *Covert channel detection* (detecção de canais secretos para roubar transmissões de dados). Em uma primeira etapa, são definidos pelo operador, parâmetros que remetam às características importantes desses serviços, gerando um perfil do serviço. Por conseguinte, datasets são usados em um otimizador bayesianos no ajuste do FMA para o funcionamento dos serviços detectáveis, considerando os perfis criados. Por fim, um programa P4 é configurado recebendo os parâmetros FMA ajustados, no qual é inserido na lógica de processamento de switches compatíveis possibilitando que essas atividades sejam detectadas durante o funcionamento da rede.

O uso de ML tem ganhado espaço em abordagens acerca da tomada de decisão, dessa maneira o estudo de (QIN et al., 2020b) propõe uma *Binary Neural Network* (BNN) no desenvolvimento de um IDS para borda de redes. A abordagem possui um controlador central na nuvem recebendo amostras do tráfego de rede de dispositivos. As amostras são usadas por este dispositivo central junto a técnica *Federated Learning* que realiza o *Backpropagation* (treinamento), ajustando os pesos de uma *Binary Neural Network* (BNN). Por outro lado, no domínio do usuário, pode ser utilizado um programa P4 executando uma BNN em switch de borda que

realiza a inferência dos pacotes no plano de dados (*Forward Propagation*). Essa técnica pode apresentar problemas a medida que os ataques variem devido à dependência do controlador externo no ajuste dos pesos, alavancado pela capacidade desse controlador analisar amostras do tráfego de rede, podendo deixar ataques passarem despercebidos. ML também é usada nos trabalhos de (PRADHAN; MANNEPALLI, 2021) e (BACHL; FABINI; ZSEBY, 2021), visto que ambos utilizam a biblioteca scikit-learn para treinar uma árvore de decisão (DT) simples no espaço do usuário, utilizando atributos básicos de conexão e uma média de tamanho dos pacotes. A DT treinada é utilizada por estes trabalhos em dois classificadores executados no espaço do usuário e Kernel, realizando a comparação de desempenho entre os dois, de tal maneira que em ambos os trabalhos o desempenho de processamento do Kernel é superior ao espaço do usuário, respectivamente em, 19% e 20%. A diferença entre os dois trabalhos é que (PRADHAN; MANNEPALLI, 2021) apresenta uma arquitetura e um estudo de vantagens de alguns algoritmos de ML. Esses dois trabalhos apresentam um algoritmo simples que processa pacotes no plano de dados através do Kernel, entretanto os trabalhos se limitam a análise de desempenho.

## 3.2 DISCUSSÃO

Conforme dados apresentados na Tabela 1, determinados trabalhos aproveitam das características dos ataques em uma tarefa de processamento externo ao plano de dados, de maneira manual ou ainda automatizada contando até mesmo com técnicas de ML na identificação das assinaturas relacionados aos ataques. As assinaturas selecionadas são convertidas para um formato compatível que funciona no plano de dados aplicando filtros aos pacotes processados. Nessa modalidade de trabalhos, a detecção é realizada no plano de controle e existe uma iteração desse controlador com o plano de dados através da escolha das assinaturas e filtros que são executadas pelos dispositivos de encaminhamento. Outra maneira de detecção externa é realizada integralmente no plano de controle, aproveitando o plano de dados somente para enviar informações necessárias para a detecção. Nesse tipo de trabalho, é comum o plano de dados coletar informações e enviá-las ao plano de controle, responsável pela detecção dos ataques.

Outros estudos, utilizam estatísticas, *Sketches* e taxas contando com um limite na identificação de ataques volumétricos diretamente no plano de dados. Esses dados focados em detecção de ataques DDoS volumétricos. Um ponto significativo ocorre após o ano de 2020, quando surgiram trabalhos que incorporam técnicas que fazem uso do ML diretamente no plano

de dados. Através do ML, essas abordagens podem reconhecer padrões normais de funcionamento da rede e conseguir detectar anomalias no seu funcionamento.

Ademais, os trabalhos em geral apresentam a verificação de um atributo selecionado pelo autor na geração do método identificador da ameaça à rede. Portanto, eles detêm a capacidade de identificar uma categoria de ataque. Por outro lado, alguns trabalhos utilizam conjuntos de dados públicos, que contêm um número maior de atributos. Isso possibilita que os testes sejam replicados para uma variedade maior de ataques e também permite que outros pesquisadores reproduzam esses testes.

Tabela 1 – Resumo de trabalhos relacionados.

| ano | Paper | ML | tipo de ataque | Detecção no plano de dados ou kernel | Detecção Realizada por | categoria (Onde é identificado o ataque) |
|---|---|---|---|---|---|---|
| 2017 | (BERTIN, 2017) | não | DDoS | não | Atributos | Detecção realizada externamente |
| 2019 | (KUKA et al., 2019) | não | DDoS | não | Atributos | Detecção realizada externamente |
| 2019 | (LEWIS; BROADBENT; RACE, 2019) | não | Diversos | não | Atributos | Configuração de regras e filtragens |
| 2020 | (ZHANG et al., 2020) | não | DDoS | não | Estatísticas | Detecção realizada externamente |
| 2020 | (CARVALHO BERTOLI et al., 2020) | não | PROBE | não | Assinaturas | Configuração de regras e filtragens |
| 2020 | (MUSUMECI et al., 2020) | sim | DDoS | Não | Atributos | Detecção realizada externamente |
| 2021 | (GRAY et al., 2021) | sim | Diversos | não | Atributos | Detecção realizada externamente |
| 2021 | (DIMOLIANIS; PAVLIDIS; MAGLARIS, 2021) | sim | Diversos | não | Assinaturas | Configuração de regras e filtragens |
| 2022 | (WANG; CHANG, 2022b) | não | Diversos | não | Assinaturas | Configuração de regras e filtragens |
| 2022 | (MUSUMECI et al., 2022) | sim | DDoS | não | Entropia | Detecção realizada externamente |
| 2022 | (ROSHANI; NOBAKHT, 2022) | sim | DDoS | não | Atributos | Detecção realizada externamente |
| 2019 | (LAPOLLI; MARQUES; GASPARY, 2019) | não | DDoS | sim | Entropia | Detecção diretamente no plano de dados |
| 2019 | (SANTOYO-GONZÁLEZ; CERVELLÓ-PASTOR; PEZAROS, 2019) | não | DDoS | sim | Entropia | Detecção diretamente no plano de dados |
| 2019 | (TAVARES; FERRETO, 2019) | não | DDoS | sim | Sketch | Detecção diretamente no plano de dados |
| 2020 | (DIMOLIANIS; PAVLIDIS; MAGLARIS, 2020) | não | DDoS | sim | Atributos | Detecção diretamente no plano de dados |
| 2020 | (SILVEIRA ILHA et al., 2020) | não | DDoS | sim | Entropia | Detecção diretamente no plano de dados |
| 2020 | (SIMSEK et al., 2020) | não | DoS | sim | Computação de Taxas | Detecção diretamente no plano de dados |
| 2020 | (KHOOI et al., 2020) | não | DDoS | sim | Sketch | Detecção diretamente no plano de dados |
| 2020 | (QIN et al., 2020b) | sim | Diversos | Sim | Atributos | Detecção diretamente no plano de dados |
| 2021 | (GAO; HANDLEY; VISSICCHIO, 2021) | não | DDoS | sim | Estatísticas | Detecção diretamente no plano de dados |
| 2021 | (DING; SAVI; SIRACUSA, 2021) | não | DDoS | sim | Entropia | Detecção diretamente no plano de dados |
| 2021 | (LIU et al., 2021) | não | DDoS | sim | Sketch | Detecção diretamente no plano de dados |
| 2021 | (DING et al., 2021) | não | DDoS | sim | Sketch | Detecção diretamente no plano de dados |
| 2021 | (BACHL; FABINI; ZSEBY, 2021) | sim | Diversos | sim | Atributos | Detecção diretamente no plano de dados |
| 2021 | (PRADHAN; MANNEPALLI, 2021) | sim | Diversos | sim | Atributos | Detecção diretamente no plano de dados |
| 2021 | (BARRADAS et al., 2021) | sim | Diversos | sim | Acumuladores | Detecção diretamente no plano de dados |
| 2022 | (GOLCHIN et al., 2022) | não | DDoS | sim | Entropia | Detecção diretamente no plano de dados |
| 2022 | (CARVALHO et al., 2022) | sim | DDoS | sim | Atributos | Detecção diretamente no plano de dados |

Fonte: Autoria própria.

# 4 ARQUITETURA PROPOSTA

A detecção de intrusões é uma área que tem evoluído constantemente com o intuito de acompanhar os ataques perpetuados por pessoas mal intencionadas, que atualmente se aproveitam da expansão de dispositivos eletrônicos, tais como, celulares, dispositivos IoT, computadores, entre outros, para efetuar a prática de crimes cibernéticos. Desse modo, conforme elencado no Capítulo 3, existe um esforço no desenvolvimento de pesquisas que buscam alternativas para otimizar o processo de detecção somado ao surgimento de novas tecnologias que possibilitam uma nova abordagem de análise, por exemplo, embutir uma nova função de identificação de anomalias em hardwares que habitualmente continham a função principal de encaminhar pacotes (*i.e.*, switches, roteadores). Esses dispositivos de encaminhamento, que serão nomeados como dispositivos alvo no restante do trabalho, operam diretamente na rede, além de estarem próximos aos equipamentos que hospedam os serviços e também dos dispositivos utilizados pelos usuários da rede, portanto são uma ótima adição de segurança.

Entretanto, esses comutadores de rede são desenvolvidos com o objetivo de processar altos volumes de tráfego de rede, ao mesmo tempo que possuem recursos limitados. Dessa forma, os meios fornecidos para uma nova atribuição de detecção de intrusões idealizada são limitados, exigindo um esforço no desenvolvimento de funções adequadas às limitações impostas pelos dispositivos alvo. Todavia, as soluções apresentadas recentemente na literatura tem se mostrado benéficas na contenção de ameaças em prol de um melhor desempenho e segurança da estrutura de rede e dos seus usuários.

A arquitetura proposta neste Capítulo tem o objetivo de fornecer um método para otimizar a detecção de intrusão, atribuindo a função de detecção em dispositivos pertencentes a infraestrutura de rede. Portanto, os resultados provenientes da análise em ambiente simulado mostram que existe a possibilidade de embutir um IDS em dispositivos de rede, enquanto o seu desempenho não é afetado de forma significativa. Por fim, através da referida arquitetura é possível gerar um modelo propício a um classificador embutido no Kernel do sistema para detecção de intrusões.

## 4.1 VISÃO GERAL

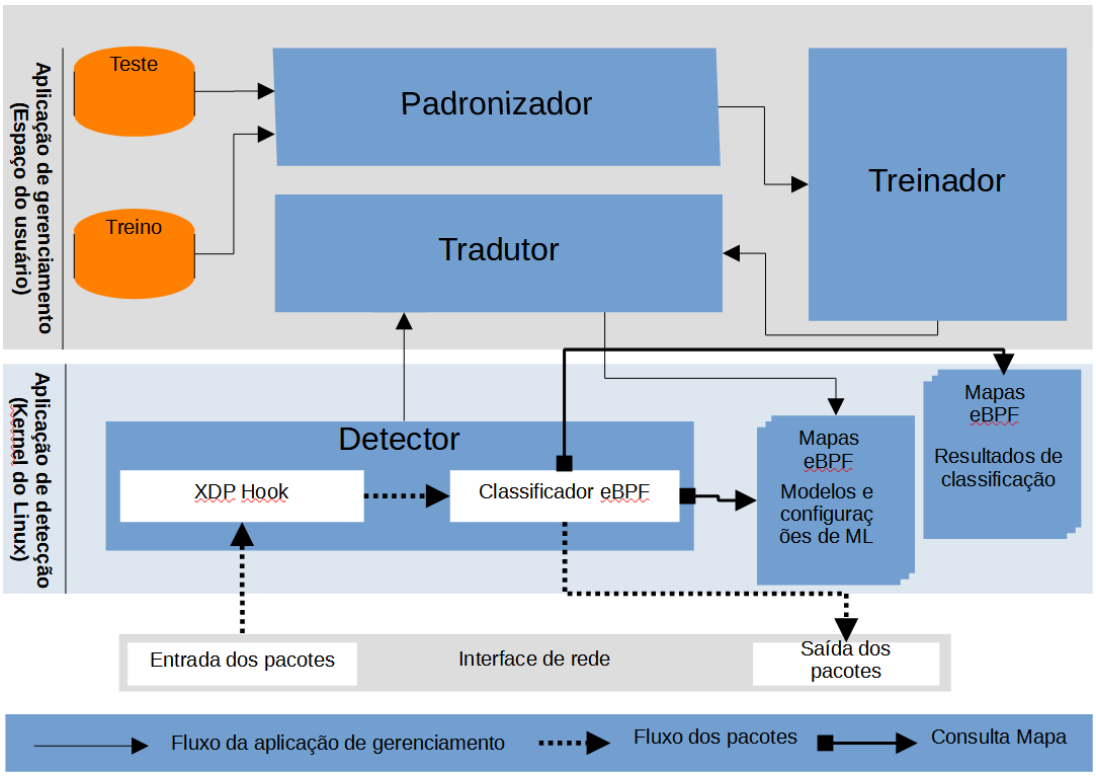
O objetivo da proposta deste Capítulo é propor uma arquitetura para um sistema de detecção de intrusões ajustável às atividades dos usuários conectados em dispositivos de rede, que pode ser usado no Kernel do sistema Linux para processamento de pacotes no plano de dados. Para tal feito, a aplicação deverá receber uma base de dados que represente o comportamento dos usuários da rede, os quais precisam ser gravados e rotulados pelo administrador para que possam ser utilizados por ML na avaliação de modelos *offline*. Em prol a atender essa demanda, foram implementadas duas aplicações, uma de gerenciamento (executada no espaço do usuário) e a segunda de detecção (executada no kernel), conforme apresentado na Figura 6. Os módulos da aplicação de gerenciamento são responsáveis pela configuração do programa executado no Kernel do sistema. Portanto, o administrador utiliza uma base de dados para treinar um classificador e gerar um modelo de ML, utilizado pelo detector.

Os atuais trabalhos disponíveis na literatura, que utilizam eBPF, detalhados na Seção 3.1, expõem apenas soluções de reconfiguração através de técnicas por assinatura, apresentando dificuldade em reconhecer novos ataques. Já os trabalhos que usam ML estão limitados a apenas avaliar a viabilidade dos processamentos de pacotes no Kernel do sistema. De acordo com o mencionado na Seção 2.2, o eBPF pode prover segurança adicionando funções em dispositivos já existentes na topologia de rede e, portanto, diminuir o custo computacional para o desempenho dessa ação. Nas próximas seções serão apresentadas com detalhes as aplicações implementadas no espaço do usuário e Kernel.

## 4.2 APLICAÇÃO DE GERENCIAMENTO

A aplicação de gerenciamento funciona no espaço do usuário em um computador que tem capacidade de suportar a execução de algoritmos de ML e seleção de atributos, além de permitir a construção de modelos otimizados para detecção de intrusões. Para tal feito, deve ser disponibilizada uma base de dados rotulada (*i.e.*, normal ou ataque), contendo os padrões de comportamento dos usuários, a partir da qual o sistema formará diversos modelos que serão analisados por um classificador com um algoritmo simplificado de ML, com o objetivo de encontrar o que tiver a melhor acurácia. No fim dessa análise, será fornecida a melhor opção de modelo do classificador no plano de dados, utilizado para a geração de um código fonte em linguagem C limitada, resultando em um programa eBPF, que será injetado no *driver* do dis-

Figura 6 – Arquitetura proposta.




Fonte: Autoria própria.

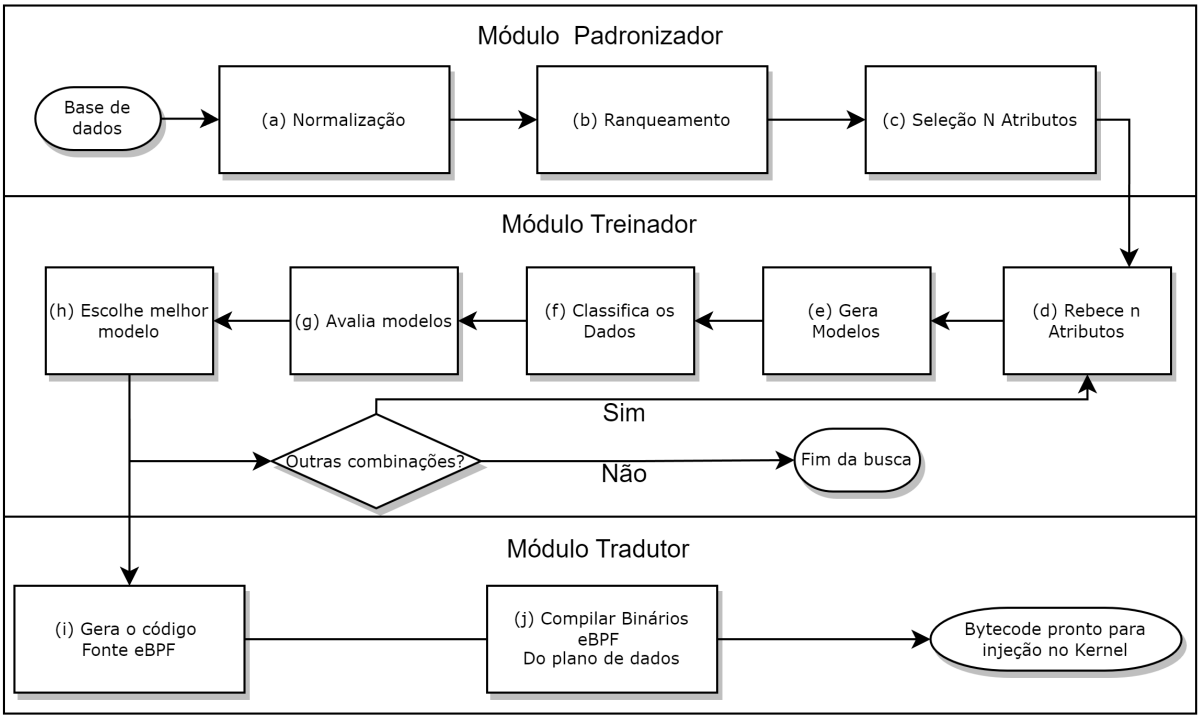
positivo de rede através de um *loader* eBPF. Os módulos da aplicação de gerenciamento são Padronizador, Treinador e Tradutor, apresentados na Figura 7, na qual os detalhes desse fluxograma são abordados a seguir:

### 4.2.1 Módulo Padronizador

O módulo Padronizador permite a entrada das bases de dados Treino e Teste fornecida pelo administrador do sistema, adequando-as aos processos envolvidos em toda a aplicação através da normalização dos números em uma escala de inteiros, que são aceitos pelos dispositivos alvo, além de realizar uma pré-seleção de atributos por ranqueamento e avaliação do conjunto de atributos por meio de três etapas descritas nesta seção.

(a) A primeira etapa do módulo é responsável por adequar a entrada dos dados conforme a necessidade do restante do sistema. Devido ao fato da referida base trazer campos gravados como *string*, que são incompatíveis com o classificador implementando, é feita a conversão dos campos descritos por caracteres em numerais, esse tratamento ocorre nos atributos Protocol_type, Service, Flag, gravados como campos descritivos. Ademais,

Figura 7 – Fluxograma da aplicação de gerenciamento.




Fonte: Autoria própria.

devido ao eBPF executado no Kernel suportar apenas números inteiros (PRADHAN; MANNEPALLI, 2021), os dados de cada coluna são convertidos em um intervalo de números entre 0 e 100. Isso é feito pelo Normalizador, que armazena o maior e menor valor de cada um dos atributos (considerando as bases de treino e teste) e aplica o método matemático de grandezas proporcionais descrito pela Equação 4.1 durante a normalização dos dados. Após a conclusão dessa etapa, uma nova base de dados é disponibilizada tendo números inteiros com um intervalo definido.

$$\frac{x}{y} = \frac{z}{k} \tag{4.1}$$

Equação 4.1 de grandezas proporcionais.

(b) A segunda etapa do módulo é executada apenas em bases de treinamento, sendo responsável por realizar o ranqueamento dos atributos, concedendo um valor de representabilidade a cada um deles em relação a uma classe, através de um cálculo estatístico e filtro, detalhados na Seção 5.2.1. Essa função deve receber o conjunto de atributos que deseja ranquear identificados pelo nome, além da quantidade que se deseja obter. É possível especificar a relação de um até quarenta e um, que é a quantidade máxima contida na base

de dados. Sendo assim, o retorno desse processamento é a relação dos melhores atributos encontrados conforme a quantidade solicitada.

(c) Durante a terceira etapa, são criadas as bases treino e teste contendo apenas os atributos ranqueados pelo processo anterior. Também é realizado um teste utilizando o algoritmo KNN, no qual a base de treinamento é considerada para formar um modelo, que tem a avaliação da acurácia usando a base de testes. Portanto, na saída do módulo são retornadas as bases de dados contendo o conjunto de atributos encontrado e o resultado da avaliação realizada.

### 4.2.2 Módulo Treinador

Esse módulo é formado por cinco etapas de funcionamento, sendo que a primeira emprega um algoritmo de busca que decide a ordem e quais conjuntos de atributos serão analisados, ademais, as etapas restantes criam, avaliam e armazenam os resultados obtidos com cada conjunto de atributos verificado.

O módulo treinador cria e avalia modelos de ML utilizando qualquer conjunto de atributos. Para isso, recebe por parâmetro duas bases rotuladas (treino e teste) e também o conjunto que ele deve processar. são realizadas diversas avaliações utilizando ML das possíveis combinações com os atributos recebidos. Desse modo, os resultados das avaliações obtidos de cada conjunto são considerados para encontrar o modelo com um número menor de amostras, atributos e acurácia. Esse módulo é formado por cinco etapas de funcionamento, sendo que a primeira emprega um algoritmo de busca que decide a ordem e quais conjuntos de atributos são analisados. Ademais, as etapas restantes criam, avaliam e armazenam os resultados obtidos com cada conjunto de atributos verificado.

$$C_{n,p} = \frac{n!}{p!(n-p)!} \tag{4.2}$$

(d) A entrada do módulo são as bases de dados utilizadas para avaliação dos modelos gerados. Os atributos contidos nessas bases são recebidos por um algoritmo encarregado de formar conjuntos de atributos usados para criação de modelos individuais, além de decidir a sequência que eles são avaliados por ML. O algoritmo é responsável por organizar a segunda etapa da seleção de atributos, que é um problema de combinações matemáticas denotadas pela Equação 4.2 contendo um número de combinações possíveis, variando

conforme a quantidade de atributos utilizados sem que ocorra repetição. Sendo assim, o algoritmo de seleção especificado controla quais conjuntos de atributos serão avaliados.

(e) Na segunda etapa, é criado um modelo com o conjunto de atributos recebido, para isso, a base de dados de treino é compactada através do agrupamento das linhas idênticas, além de contabilizar ocorrências iguais em classes distintas para que se obtenham as porcentagens de cada amostra separada por classes. Ao fim do processo, os dados são expostos em uma base de classificação binária, na qual pode ser definido que amostras com porcentagens acima de 51% serão mantidas ou, ainda, definir limiares maiores. Essa organização de linhas únicas, separadas por classes com maior representabilidade, forma um modelo com os padrões de tráfego mais relevantes, consumido pelo classificador.

(f) O modelo criado é utilizado por um classificador que executa uma implementação simplificada do algoritmo KNN nomeada como L-KNN, sendo empregado em duas fases de funcionamento do programa. Na primeira, é executada durante a avaliação dos modelos criados por esse módulo onde a detecção é feita pelo classificador, realizando a predição de dados providos da base de dados. Na segunda, é efetuada durante uma aferição de desempenho, classificando os dados em tempo real na rede com fins de testes do Detector no Kernel, através de experimentos realizados com tráfego real, abordados no Capítulo 6. Sendo assim, são classificadas as amostras oriundas da base de dados de teste, tendo como saída uma lista com as predições dos dados fornecidos.

(g) As predições obtidas pelo Classificador são processadas por uma função de avaliação, que as compara com a Classificação original da base de dados. Desse modo, os resultados da classificação original e também a predição do Classificador são comparados utilizando a métrica acurácia, que avalia a capacidade do Classificador de identificar padrões de comportamento através do presente modelo. Através dessa etapa é obtido o resultado da avaliação do modelo, que será adicionado em uma lista.

(h) A última etapa do módulo Treinador recebe uma lista como entrada e decide qual o melhor modelo entre as avaliações que ela contém. A lista com os dados da avaliação dos modelos formados por conjuntos de atributos é ordenada pela maior acurácia e menor quantidade de atributos, de tal modo que o melhor modelo ficará no topo sendo mantido a cada iteração. Todo esse processo é realizado devido ao fato do KNN ser um algoritmo *lazy* e não possuir uma representação compacta dos objetos. Portanto, a predição

pode ter um grande esforço computacional em um conjunto grande de dados. Por isso, é importante a seleção de atributos relevantes ao problema relatado, diminuindo o custo computacional da classificação para que seja gerado um modelo que pode ser usado pelos classificadores de espaço do usuário e também do Kernel para processamento de pacotes na rede. Portanto, a cada conjunto de atributos avaliado, o módulo retorna o melhor modelo e classificador sempre que encontrar um modelo melhor aos anteriormente avaliados.

### 4.2.3 Módulo Tradutor

O último módulo da aplicação de gerenciamento irá gerar o código fonte em linguagem C limitada de um programa eBPF, baseado nas informações contidas no modelo e algoritmo de ML recebido do treinador. Nesse processo são geradas as partes variáveis do classificador de Kernel embutidas no meio do código fonte do programa, incluindo o modelo que também é gravado. As partes que recebem códigos variáveis são o classificador e a declaração do mapa eBPF (modelo). O código gerado é compilado em um programa eBPF, que será injetado pelo *loader* eBPF no *driver* do dispositivo de rede.

(i) Para tal feito, o módulo deve manusear o modelo recebido que é usado pelo algoritmo L-KNN empregado no espaço do usuário, disponibilizando-o em um código compatível com os dispositivos alvo. Esse modelo deverá conter o nome dos atributos desejados e a classe dos dados, essas informações são utilizadas para personalizar os códigos fonte do classificador, declaração de cabeçalhos e modelo que são utilizados pela Aplicação de Detecção. A saída desse componente é formada por códigos fonte compatíveis com o compilador eBPF.

(j) A segunda etapa do módulo é responsável por compilar o código gerado e verificar se pode ser usado e embutido no dispositivo Alvo. Sendo assim, essa função aciona o LLVM CLang que compila o código fonte criado e verifica a sanidade do programa eBPF obtido, isto é feito para garantir que o programa execute corretamente, através de uma simulação, verificando se memória e registradores estão sempre em estado válido até o término do programa executado. Além disso, também é verificado se as chamadas aos *helpers* são permitidas e, caso não ocorra nenhum erro, o programa eBPF estará pronto para ser inserido no Kernel do sistema com o intuito de classificar o tráfego de rede, utilizando o XDP

*hook* na interceptação dos pacotes analisados (MIANO et al., 2018) com auxílio do mapa eBPF gerado pelo treinador para realizar as classificações.

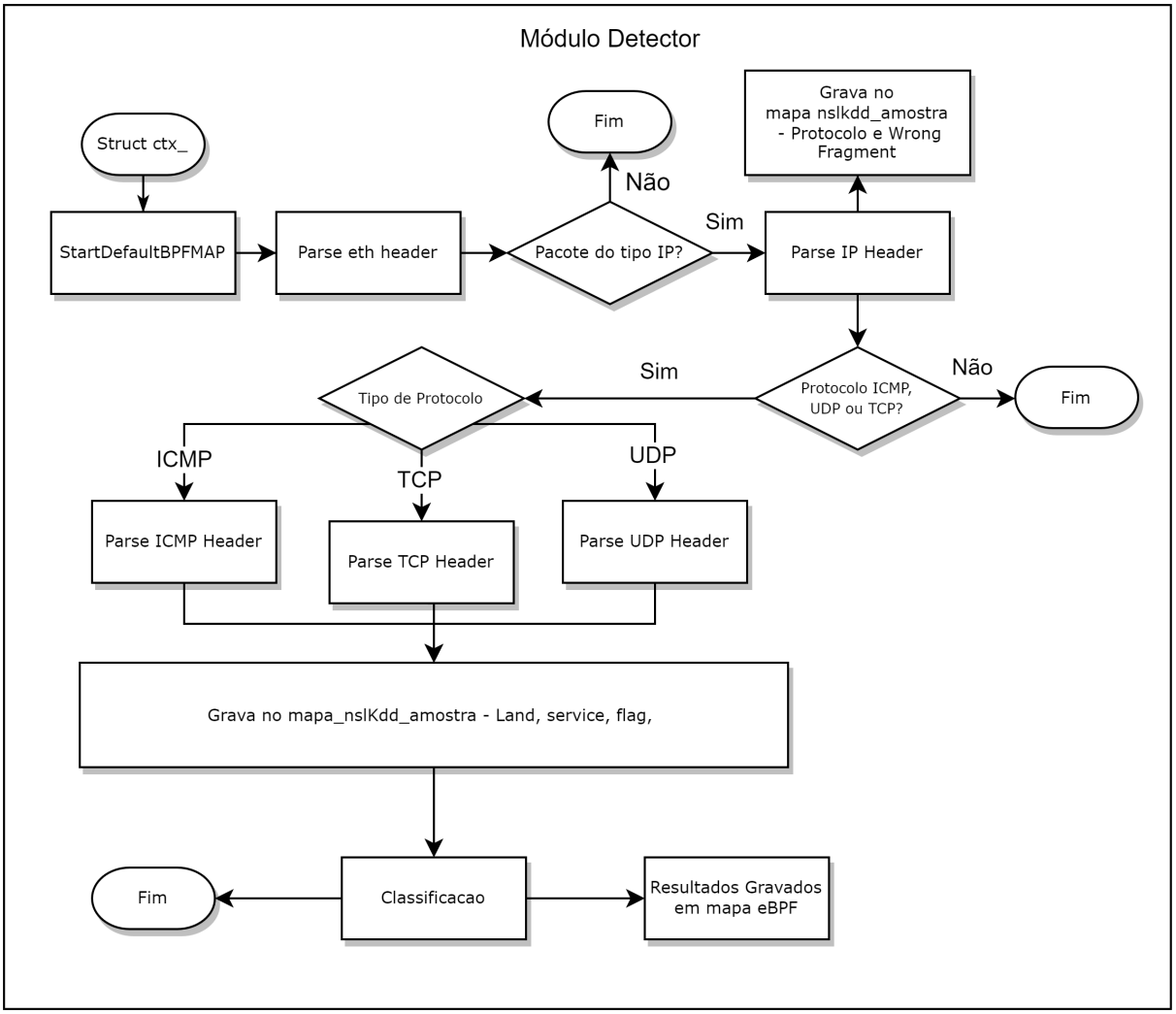
## 4.3 APLICAÇÃO DE DETECÇÃO

A aplicação de Detecção, por sua vez, tem o objetivo de analisar os pacotes na rede e definir, através de um algoritmo de ML, se o tráfego é anômalo ou normal. Essa aplicação funciona diretamente em dispositivos de rede, pois analisa os dados da rede em tempo real, além de ser provida da capacidade de processar altas demandas de pacotes por segundo. O fato de operar no dispositivo de rede proporciona os meios de processamento específicos que são requisitos para alcançar um processamento com alto desempenho (CEROVIĆ et al., 2018). Essa aplicação é mostrada na Figura 8, contendo um fluxograma composto por um módulo Detector e mapas eBPF, onde são armazenadas as informações necessárias para o seu funcionamento

A aplicação é injetada no Kernel de um dispositivo alvo com suporte à tecnologia eBPF, recebendo um modelo usado por ML construído pela aplicação de gerenciamento (gravados em um mapa eBPF), para efetuar a análise dos pacotes recebidos. Além do modelo, o algoritmo de ML também deverá ser fornecido pela aplicação de gerenciamento ao módulo detector executado no Kernel, devendo-se tomar cuidado para que esse algoritmo funcione com o modelo fornecido. Uma vez que o dispositivo de rede tenha o programa eBPF funcionando, a extração dos dados do tráfego de rede real é realizada pelo *hook* XPD, que captura os pacotes da rede os enviando ao classificador, responsável por analisar os cabeçalhos (IP, TCP, ICMP, UDP) dos dados recebidos, classificando-os em normal ou ataque e, baseado nessa classificação, o eBPF pode realizar ações como permitir que o pacote prossiga, descartar ou, ainda, redirecionar esse pacote. As amostras provindas da rede são comparadas àquelas gravadas no mapa eBPF fornecido pelo módulo Treinador. A classificação do tráfego de rede fica a cargo do algoritmo L-KNN, que realizará uma comparação dos pacotes com os dados presentes no modelo disponível. A aplicação implementada para funcionar no Kernel do sistema é um detector que classificará os dados da interface de rede conforme os padrões estabelecidos em seu modelo (mapa eBPF). Seu funcionamento será detalhado a seguir:

O módulo Detector da aplicação de Kernel é um programa eBPF executado no *driver* da interface de rede do dispositivo alvo, tendo o seu funcionamento disparado através de um XDP *hook*, que realiza a interceptação individual de pacotes diretamente nessa interface. Através dos pacotes capturados é possível realizar a leitura dos cabeçalhos a fim de obter as informações

Figura 8 – Fluxograma da aplicação Kernel do Linux




Fonte: Autoria própria.

necessárias para a formação dos atributos, que serão utilizados na classificação do tráfego de rede. As informações obtidas são usadas para criar atributos através de algum processamento que pode ser uma simples leitura, contagem ou cálculos matemáticos. Os atributos são utilizados em uma comparação com os que estão gravados em um modelo disponível num mapa eBPF que, por sua vez, armazena na memória do dispositivo alvo o modelo de ML, criado na Aplicação de Gerenciamento. Entretanto, nada impede que o administrador do sistema altere o mapa eBPF modificando, adicionando ou removendo linhas. Após a extração, os atributos são comparados com os dados armazenados no modelo por um classificador, que grava o resultado da classificação em um outro mapa eBPF, visível ao administrador do sistema.

# 5 IMPLEMENTAÇÃO

Para validar a arquitetura apresentada no Capítulo 4, um protótipo das Aplicações de Gerenciamento e Detecção foi implementado. A Aplicação de Gerenciamento é responsável pela configuração do programa eBPF utilizado pela Aplicação de Detecção, empregando a linguagem de programação Python. Enquanto a Aplicação de Detecção é um classificador no Kernel para processamento de pacotes na rede, escrito com a linguagem de programação C restrita para eBPF. Nesse capítulo, será descrita a maneira com a qual os módulos integrantes dessas aplicações são implementados.

.

## 5.1 COMUNICAÇÃO ENTRE MÓDULOS E COMPONENTES

Os três módulos principais da aplicação de gerenciamento são programas individuais que aguardam uma entrada para gerar uma saída útil ao módulo seguinte. As entrada dos parâmetros de cada módulo pode ser acionada por terminal de comando executado pelo módulo anterior, um breve resumo do funcionamento é detalhado abaixo:

Na entrada da Aplicação de Gerenciamento, devem ser informadas as bases de dados, a lista de atributos que se deseja avaliar e a quantidade de atributos que serão ranqueados. Sendo assim, o módulo padronizador retorna as bases de dados com a quantidade de atributos selecionados por ranqueamento e a avaliação da acurácia, *F1-Score*, *recall* e precisão deles. Nesse ponto, o administrador pode escolher prosseguir ao próximo módulo selecionando as bases de treino, teste e uma métrica para a avaliação (*i.e.*, Acurácia, *F1-Score*), a partir desse ponto, o módulo realiza diversas avaliações em subconjuntos de dados, repassando o melhor modelo ao módulo tradutor para criar códigos fontes em linguagem C limitada. Os códigos fontes gerados são compilados em um programa eBPF compatível com os dispositivos alvos, uma vez compilado o módulo pode acionar o *loader* através de terminal de comando para embutir esse programa no Kernel.

Já o módulo Detector da aplicação no Kernel recebe um programa eBPF injetador pelo um *loader*. Uma vez injetado no dispositivo alvo, o *hook* XDP interceptará cada pacote que atravessa a interface de rede, analisando cada um dos cabeçalhos em sequência para obter os atributos usados na classificação do tráfego de rede.

## 5.2 APLICAÇÃO DE GERENCIAMENTO

A aplicação de Gerenciamento implementada manipula os dados recebidos na intenção de fornecer ao classificador no Kernel um algoritmo e também um modelo. Os módulos implementados nessa seção são mostrados no fluxograma exibido na Figura 7 contendo os módulos Padronizador, Treinador e Tradutor.

### 5.2.1 Módulo Padronizador

A primeira etapa da solução é aplicada pelo módulo Padronizador, que tem a implementação das funções de normalização, ranqueamento e escolha do melhor conjunto de atributos que serão apresentados a seguir.

(a) Inicialmente, as bases de dados são recebidas pela função de normalização que faz a conversão de *strings* para números, essa conversão é realizada através da função do Python Pandas Categorical (pd.Categorical), categorizando as *strings* em uma lista com elementos únicos, sendo organizados em numerais, para que seja considerada a numeração obtida da categoria com o comando (categoria.cat.code) que obtém o índice da lista formada. Uma outra função do Normalizador é escalar cada atributo da base de dados em um intervalo numérico definido. Sendo assim, o código fonte da função de grandezas proporcionais responsável por fazer a conversão pode ser observado na Figura 9, onde são recebidos o v_maximo (maior valor na coluna), a escala (valor máximo a ser considerado nesse caso 100) e o v_converter (valor a ser convertido). Alguns exemplos de conversão podem ser observados na Tabela 2.

Figura 9 – Função Python de normalização de valores entre 0 e 100.

```
def regrade3_v2(v_maximo, v_escala, v_converter,)
    retorno = v_escala * v_converter//v_maximo    ## 200/2
```

Fonte: Autoria própria.

(b) Após os dados serem adequados, ocorre a primeira fase de seleção de atributos elaborada durante esse módulo através de ranqueamento. Na seleção de atributos existem duas abordagens principais para reduzir a dimensionalidade dos dados: sendo elas, transformação de atributos, onde os dados são mesclados formando novos atributos e, também,

Tabela 2 – Exemplos de conversão em valores de atributos, exibidos na saída, convertidos conforme a escala referente ao valor máximo.

| **Atributo** | **V. maximo** | **V. Escala** | **V. Converter** | **Saída** |
|---|---|---|---|---|
| Protocolo | 2 | 100 | 0 | 0 |
| Protocolo | 2 | 100 | 1 | 50 |
| Protocolo | 2 | 100 | 2 | 100 |
| Service | 71 | 100 | 10 | 14 |
| Service | 71 | 100 | 51 | 72 |
| Service | 71 | 100 | 71 | 100 |

Fonte: Autoria própria.

seleção de atributos, em que são escolhidos atributos relevantes na representação dos dados. Ambas abordagens são utilizadas em técnicas de mineração de dados na identificação dos atributos mais relevantes, aumentando a capacidade de predição do classificador e também a elaboração de um modelo de menor tamanho, que possibilita um esforço computacional inferior. Com vistas a simplificar o processo de avaliação foi escolhida a abordagem de seleção de atributos para a elaboração dos experimentos. Os principais métodos de seleção dos atributos são *filter* (filtros), *wrappers* (Envolucro) e *Embedded* (embutido) (KHAMMASSI; KRICHEN, 2017). Nesse módulo, a estratégia de filtro e o método chi-square são empregados. Em suma, filtros podem avaliar cada atributo individualmente em relação a uma classe, com auxílio do ranqueamento realizado por alguma estratégia de heurística. Nesse ponto, o método chi-square viabiliza a estimativa dos atributos através de um teste estatístico. Sendo assim, a importância de cada atributo pode ser ranqueada.

(c) Na última etapa do módulo é feita a escolha e a avaliação de um subconjunto de atributos usado na segunda etapa de seleção elaborada pelo módulo Treinador. A fim de obter um modelo mais preciso referente a um tipo de ataque, os dados do NSL-KDD são personalizados de tal modo que se mantém apenas as linhas que endossam as classes normal e ataques DDoS na seleção de um conjunto de atributos escolhidos com auxílio da função de ranqueamento. Essa base de dados DDoS é usada como entrada no módulo de padronização e testada em diferentes configurações de atributos.

Na escolha desse subconjunto de atributos, são usadas como entrada listas de diferentes

conjuntos de atributos e também a quantidade ranqueada. Dessa maneira, em uma primeira avaliação, obtém-se os melhores resultados com todos os 41 atributos presentes, para se definir um ponto de partida de comparação com o restante dos testes, realizados com uma menor quantidade de atributos. Em um segundo momento, são removidos 13 atributos relevantes em HIDS mostrados por (DHANABAL; SHANTHARAJAH, 2015) na Tabela 3 do artigo deste autor, restando 28 atributos. Os atributos removidos contabilizam informações como tentativas de acesso a sistemas e diretórios, portanto são de difícil acesso a um NIDS. Na terceira seleção, são considerados 9 atributos básicos de conexão descritos na Tabela 2 do trabalho de (DHANABAL; SHANTHARAJAH, 2015), sendo esses delimitados para a área de seleção para ranqueamento, escolhidos devido a sua facilidade de obtenção, por não necessitarem guardar dados em períodos de tempo e também a sua capacidade de predição não ser afetada de maneira significativa. Desse conjunto de 9 atributos foram ranqueados 6, essa decisão foi tomada devido a esse número conseguir representar ataques DDoS e Probe sem grande detrimento do desempenho, alegado por (LIN; KE; TSAI, 2015) e (DHANABAL; SHANTHARAJAH, 2015). Entretanto, foi realizado mais um teste com apenas 5 atributos, sendo possível verificar que a predição não seria afetada pelo atributo Duration, pois seria de difícil implementação no Kernel e, portanto, decidiu-se definir que apenas 5 atributos seriam selecionados ao componente treinador. Os resultados dessas avaliações são expressos na Tabela 3, ademais, a relação dos grupos estão na listagem a seguir, descritos pela palavra conjunto seguida pelo número de atributos utilizados:

- Conjunto 41 - duration, protocol_type, service, flag, src_bytes, dst_bytes, land, wrong_fragment, urgent, hot, num_failed_logins, logged_in, num_compromised, root_shell, su_attempted, num_root, num_file_creations, num_shells, num_access_files, num_outbound_cmds, is_host_login, is_guest_login, count, srv_count, serror_rate, srv_serror_rate, rerror_rate, srv_rerror_rate, same_srv_rate, diff_srv_rate, srv_diff_host_rate, dst_host_count, dst_host_srv_count, dst_host_same_srv_rate, dst_host_diff_srv_rate, dst_host_same_src_port_rate, dst_host_srv_diff_host_rate, dst_host_serror_rate, dst_host_srv_serror_rate, dst_host_rerror_rate, dst_host_srv_rerror_rate.
- Conjunto 28 - duration, protocol_type, service, flag, src_bytes, dst_bytes, land, wrong_fragment, urgent, count, srv_count, serror_rate, srv_serror_rate, rerror_rate,

srv_rerror_rate, same_srv_rate, diff_srv_rate, srv_diff_host_rate, dst_host_count, dst_host_srv_count, dst_host_same_srv_rate, dst_host_diff_srv_rate, dst_host_same_src_port_rate, dst_host_srv_diff_host_rate, dst_host_serror_rate, dst_host_srv_serror_rate, dst_host_rerror_rate, dst_host_srv_rerror_rate.

- Conjunto 9 - duration, protocol_type, service, flag, src_bytes, dst_bytes, land, wrong_fragment, urgent.
- Conjunto 6 - duration, protocol_type, service, flag, land, wrong_fragment
- Conjunto 5 - protocol_type, service, flag, land, wrong_fragment

Tabela 3 – Avaliação dos conjuntos de atributos avaliados na primeira seleção.

| **Atributos** | **F1-Score** | **Recall** | **Precisão** | **Acurácia** |
|---|---|---|---|---|
| Conjunto_41 | 89,58% | 82,40% | 98,13% | 90,60% |
| Conjunto_28 | 86,15% | 76,91% | 97,90% | 87,82% |
| Conjunto_9 | 89,09% | 81,62% | 98,07% | 90,19% |
| Conjunto_6 | 89,06% | 81,58% | 98,05% | 90,17% |
| Conjunto_5 | 88,77% | 81,10% | 98,04% | 89,93% |

Fonte: Autoria própria.

### 5.2.2 Módulo Treinador

Após a definição do subconjunto inicial, capaz de representar a categoria de ataque DDoS, esse tipo de ataque é escolhido com o objetivo de elaborar um modelo de classificação específico. Portanto, essa categoria de ataques é escolhida afim de se obter avaliações da base de dados realizadas, nas quais é selecionado o modelo utilizado no Kernel. Nessa etapa, executada pelo módulo Treinador, ocorre uma segunda fase de seleção de atributos com a finalidade de avaliar e escolher a melhor configuração do classificador, que pode ser usado na identificação do ataque.

(d) O grupo de atributos e a base de dados obtidos a partir da etapa anterior são avaliados pelo módulo de Treinamento, através de um ciclo de instruções para seleção de um modelo de ML, utilizando um algoritmo que decide quais conjuntos de atributos serão avaliados. A seleção de atributos executada pode ser generalizada como um problema de otimização combinatória (FEO; RESENDE, 1989), nesse ponto a técnica GRASP de (QUINCOZES

et al., 2020) pode localizar o conjunto de atributos mais rapidamente através de uma estratégia com filtros internos. Problemas dessa modalidade podem ser resolvidos com o uso de meta-heurísticas como GRASP (QUINCOZES et al., 2021), uma meta-heurística híbrida, que combina técnicas de filtragem com a finalidade de reduzir o número de atributos a serem analisados posteriormente por um procedimento baseado em *wrapping* com suporte de algoritmos de ML e possui vantagens em cima de outras metodologias como busca sequencial (QUINCOZES et al., 2020). *Wraper* é uma técnica de seleção que realiza a aferição (*i.e.*, acurácia, F1-Score) em vários subconjuntos de atributos, viabilizando a escolha do melhor. Ademais, com a finalidade de informação, a seleção baseada no método *embbedded* é específica de alguns algoritmos de classificação, nesse tipo de abordagem, o subconjunto de atributos é selecionado durante a geração do modelo de classificação no decorrer do treinamento (*i.e.*, Árvores de Decisão).

(e) Na segunda etapa do módulo, é criado um modelo com cada conjunto de atributos recebido. Para isso, os dados são agrupados em linhas únicas separadas por classe. Além disso, uma nova linha é adicionada contendo a quantidade de ocorrência de dados iguais. No final do processo, obtém-se uma nova base de dados contendo apenas linhas únicas com atributo(s), classe e quantidade de ocorrência da linha em questão. Tais dados são ordenados de modo decrescente, seguindo os critérios quantidade e classe. Portanto, espera-se uma tabela organizada, em que a maior quantidade de ocorrências de um padrão esteja na parte superior e a menor quantidade logo abaixo. Por fim, apenas os atributos com maior quantidade são usados na elaboração de um modelo, sendo também removida a coluna quantidade. Um exemplo da saída dos dados é mostrado na Tabela 4.

(f) O modelo gerado é entregue ao classificador L-KNN, observando que, por tratarem-se de duas aplicações distintas (executadas no espaço do usuário e Kernel), decidiu-se pela utilização do mesmo algoritmo nos classificadores de ambas e, portanto, as funções de treinamento e classificação obedecem a mesma lógica de processamento, mesmo que o classificador implementado no espaço do usuário utilize a linguagem de programação Python e o classificador do Kernel tenha sido escrito em linguagem C limitada para Kernel eBPF. No entanto, portar o algoritmo KNN da biblioteca Scikit learn (SCIKIT-LEARN, 2022) escrito em Python para linguagem eBPF demandaria a conversão de diversas funções e classes, incluindo algumas funções de outras bibliotecas como Numpy, utilizada para armazenar os dados em uma lista. Devido às limitações impostas a um programa

executado no Kernel do sistema, optou-se por implementar uma versão própria e mais simples do algoritmo KNN (L-KNN) na execução do experimento descrito no trabalho em questão. O pseudocódigo do algoritmo pode ser visualizado em Algoritmo 1. Basicamente o algoritmo implementado e utilizado nos testes realizados faz uma comparação de distâncias entre amostras extraídas dos pacotes com as amostras do modelo (Linhas 4-9) e armazena as menores distâncias e a classificação para esses resultados. Ao final de todas as comparações realizadas, são preservadas as classificações das menores distâncias entre a amostra e os dados (Linhas 10-15) e a decisão é tomada através de votação majoritária. No que tange a comparação de distâncias do algoritmo KNN, empregou-se a distância *manhattan*, denotada na Equação 5.1, devido às operações matemáticas do eBPF serem limitadas a soma, subtração, multiplicação, divisão e resto (VIEIRA et al., 2019). Um exemplo de funcionamento do algoritmo L-KNN Implementado pode ser observado na Tabela 5, conforme pode ser visualizado, cada coluna da amostra é subtraída da coluna equivalente ao mesmo tipo de dado do modelo armazenado e os resultados das subtrações originam as distâncias computadas, sendo considerada(s) a(s) menor(es) aquela(s) que for(em) mais próxima(s) de 0.

$$D = \sum_{i=1}^{n} |x_i - y_i| \tag{5.1}$$

**Algoritmo 1:** IMPLEMENTAÇÃO SIMPLIFICADA DO KNN (L-KNN)

**Entrada:** Um conjunto $amostras_{treino}$ com $n$ assinaturas com $m$ atributos cada, o número $K$ de vizinhos mais próximos e uma amostra $t$ cuja classe é desconhecida.
**Saída:** As $K$ amostras vizinhas mais próximos de $t$.

```
1  início
2      KNN ← ∅ // inicialização dos K vizinhos mais próximos
3      para cada a ∈ amostras_treino faça
4          dist_a_t ← 0 // distância Manhattan entre a e t
5          numAtributos ← |t| // quantidade de atributos de t
6          enquanto i < numAtributos faça
7              dist_a_t ← |atributo_a[i] − atributo_t[i]| + dist_a_t
8              i++
9          fim
10         para cada k ∈ 1...K faça
11             se dist_a_t < dist_k_t então
12                 KNN_k ← a // a amostra a é a k^th mais próxima de t
13                 fim para // interrompe o laço e não afeta os k maiores
14             fim
15         fim
16     fim
17 fim
18 retorna KNN // as K amostras mais próximos de t
```

(g) Após a classificação dos dados ser concluída, uma relação com as predições obtidas através do classificador é comparada com a classificação original dos dados nessa etapa. Essa

Tabela 4 – Saída de dados processados pelo treinador, com destaque em cinza escuro das linha que são mantidas na elaboração do modelo.

| protocol_type | service | flag | wrong_fragment | classe | classe Sum | Percent |
|---|---|---|---|---|---|---|
| 0 | 20 | 90 | 0 | 0 | 4089 | 89,16 |
| 0 | 20 | 90 | 0 | 1 | 497 | 10,84 |
| 0 | 21 | 90 | 0 | 0 | 2689 | 93,40 |
| 0 | 21 | 90 | 0 | 1 | 190 | 6,60 |
| 0 | 21 | 90 | 33 | 0 | 198 | 100,00 |
| ... | ... | ... | ... | ... | ... | |
| 50 | 26 | 20 | 0 | 1 | 7 | 100,00 |
| 50 | 26 | 30 | 0 | 0 | 2 | 100,00 |
| 50 | 26 | 40 | 0 | 0 | 5 | 71,43 |
| 50 | 26 | 40 | 0 | 1 | 2 | 28,57 |
| 50 | 26 | 50 | 0 | 0 | 1103 | 100,00 |

Fonte: Autoria própria.

comparação permite a avaliação do modelo, realizada por meio dos cálculos de acurácia, F1-Score, *recall* e precisão, sendo que os detalhes de como o cálculo é realizado podem ser consultados na seção 6.1.2. O resultado da avaliação e os atributos utilizados são repassados para a próxima etapa.

(h) Por meio da avaliação recebida são obtidos os valores de acurácia, quantidade e a descrição dos atributos utilizados na formação do modelo. Esse resultado é adicionado em uma lista com a ordenação baseada no valor da maior acurácia e menor quantidade de atributos. Desse modo, são obtidos os valores dos melhores modelos ao classificador, que está no topo da lista para ser enviado ao próximo módulo da aplicação.

### 5.2.3 Módulo Tradutor

A última etapa é realizada pelo módulo Tradutor criando um código fonte C para eBPF, baseado no modelo escolhido pelo Treinador e o compila, gerando um programa compatível com os dispositivos que suportem a tecnologia.

(i) Para essa tarefa, é realizada a leitura do modelo escolhido, que será usado na criação de códigos fontes personalizados inseridos na declaração do cabeçalho e inicialização do

Tabela 5 – Comparação de distância da amostra (parte superior central), com os dados armazenados no modelo visíveis nas colunas à esquerda, além da distância entre amostra\modelo estar localizada nas colunas a direita.

| **Nova Amostra** | | | | |
|---|---|---|---|---|
| **protocol-type** | **service** | **flag** | **wrong-fragment** | **classe** |
| 50 | 27 | 60 | 0 | ? |

| **Dados Presentes no modelo** | | | | | **Distância calculada** | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **protocol type** | **service** | **flag** | **wrong fragment** | **classe** | **protocol type** | **service** | **flag** | **wrong fragment** | **Distância** |
| 0 | 20 | 90 | 0 | 0 | 50 | 7 | 30 | 0 | 87 |
| 0 | 21 | 90 | 0 | 0 | 50 | 6 | 30 | 0 | 86 |
| 0 | 21 | 90 | 33 | 0 | 50 | 6 | 30 | 33 | 119 |
| 0 | 72 | 90 | 0 | 1 | 50 | 45 | 30 | 0 | 125 |
| 0 | 89 | 90 | 0 | 1 | 50 | 62 | 30 | 0 | 142 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 50 | 27 | 50 | 0 | 0 | 0 | 0 | 10 | 0 | 10 |
| 50 | 27 | 60 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 50 | 27 | 70 | 0 | 1 | 0 | 0 | 10 | 0 | 10 |
| 50 | 27 | 90 | 0 | 1 | 0 | 0 | 30 | 0 | 30 |
| 50 | 27 | 100 | 0 | 0 | 0 | 0 | 40 | 0 | 40 |

Fonte: Autoria própria.

mapa eBPF. No código do cabeçalho são inseridos atributos e classe obtidos pelo nome de cada coluna do modelo, no qual é declarada uma estrutura demonstrada na Figura 10, contendo o formato dos dados que serão lidos pelo classificador. Também é preenchida a declaração de uma constante informando o tamanho máximo do mapa, demonstrada no código eBPF disponível na Figura 11. Tanto a estrutura quando a declaração do mapa (modelo) são utilizadas sempre que necessário ler ou gravar na memória através de um ponteiro. Um exemplo do uso da estrutura para acesso aos dados ocorre na Figura 12. Por fim, no código fonte do classificador são inseridos os atributos presentes no modelo, um exemplo pode ser observado na Figura 13.

(j) O código fonte personalizado pela primeira etapa do módulo precisa ser compilado por um compilador LLVM CLANG, para gerar o programa inserido no dispositivo alvo. Essas tarefas são realizadas, respectivamente, por linha de comando utilizando o compilador e o *Loader* eBPF. Caso não ocorra nenhum erro ou rejeição pelo verificador JIT, o novo classificador estará no dispositivo de rede.

Figura 10 – Declaração da estrutura de atributos selecionados.

```
struct knn_features_selecionadas {
        ////Atributos e classes gerados pelo MakeBPF
        __u64 protocol_type ;
        __u64 service ;
        __u8 classe ;

};
```

Fonte: Autoria própria.

Figura 11 – Declaração do mapa eBPF usado para gravar os atributos extraídos utilizando a estrutura knn_features_selecionadas na gravação dos dados.

```
struct bpf_map_def SEC("maps") nslkdd_mapa_selecionado = {
  .type        = BPF_MAP_TYPE_ARRAY,
  .key_size    = sizeof(__u32),
  .value_size  = sizeof(struct knn_features_selecionadas),
  .max_entries = LinhasModelo, //Declaracao de Constante gerada
     automaticamente pelo MakeBPF
};
```

Fonte: Autoria própria.

## 5.3 APLICAÇÃO DETECÇÃO

A aplicação de detecção implementada obedece a lógica de processamento determinada pela aplicação do gerenciamento. Entre as tarefas realizadas pela aplicação, incluem a extração de atributos diretamente do plano de dados, a comparação dos atributos com mapas eBPF (modelo), a classificação dos pacotes em ataques ou normal e a gravação dos resultados da classificação. A lógica de funcionamento da aplicação do Kernel (detector) é demonstrada no fluxograma da Figura 8. Em resumo, os *parsers* XDP realizam a extração dos dados diretamente na interface de rede através das análises e gravações de informações necessárias para a classificação dos dados nos mapas eBPF. Após percorrer todos os cabeçalhos analisáveis, a classificação é realizada conforme a comparação com o modelo gravado.

O programa eBPF é executado em cada pacote que atravessa a interface de rede, recebendo por parâmetro uma estrutura xdp_md, contendo o registro do endereço de memória inicial e final do pacote. Neste aspecto, o fluxograma representado pela Figura 8 demonstra o funcionamento da aplicação de Kernel, tal que os cabeçalhos são analisados nesta sequência. A leitura é feita de maneira semelhante em cada cabeçalho, seguindo o processo em que o atual é analisado com base no endereço inicial recebido pelo ponteiro, utilizado para definir o início e

Figura 12 – Inicialização do mapa eBPF com os atributos fornecidos.

```
static __always_inline
__u32 nslkdd_mapa_selecionadostartMap()
{
  __u32 key=0;

  struct knn_features_selecionadas *rec;
  rec = bpf_map_lookup_elem(&nslkdd_mapa_selecionado, &key );
  if(!rec)
    return XDP_ABORTED;
        // atributos e classes sao geradas pelo MakeBPF baseadas no
           modelo recebido
  rec->protocol_type=100;
  rec->service=62;
  rec->classe=1;
        ...
        ...
```

Fonte: Autoria própria.

o fim do cabeçalho atual e também em uma verificação de segurança para impedir que ocorra alguma leitura fora da área, ou ainda, do pacote demarcado pelo endereço final. Caso essa verificação seja bem sucedida, são feitas as extrações dos campos pertencentes ao cabeçalho atual, empregado na formação dos atributos. Logo após, é somado o tamanho desse cabeçalho ao seu primeiro endereço de memória, indicando a posição de início do próximo a ser analisado. O processo é repetido por cada um dos *parsers*, até que o último cabeçalho seja extraído.

O primeiro cabeçalho analisado é o ethernet, no qual é verificado se o pacote é do tipo IP. Caso afirmativo, a leitura do cabeçalho IP obtém diversas informações (iph tos,iph tot_len,iph-

Figura 13 – Cálculo de distância Manhattan no classificador L-KNN.

```
static __always_inline __u32 xdp_classificador( ){
...
struct  knn_features_selecionadas  *rec  = bpf_map_lookup_elem(&
   nslkdd_mapa_selecionado, &chave  );
        if (!rec)
          return XDP_ABORTED;
                //// Atributos gerados pelo MakeBPF no classificador,
                    conforme modelo recebido.
        aux->distancia=manhattan(rec->protocol_type,protocol_type);
        aux->distancia+=manhattan(rec->service,service);
...
```

Fonte: Autoria própria.

id,iph frag_off,iph ttl,iph protocol,iph check,iph saddr,iph daddr) gravadas em um mapa eBPF auxiliar para posterior uso das demais funções do sistema, visto que o mapa eBPF fica disponível durante toda a execução do programa. Os atributos protocolo e *wrong fragment* (número total de fragmentos incorretos nessa conexão) são coletados nesse momento, embora *wrong fragment* necessite do processamento adicional implementado no código visível na Figura 14 para verificar se ocorreram erros de fragmentação na conexão, ademais, o atributo protocolo é usado para indicar a próxima leitura.

Figura 14 – Código fonte da extração do atributo Wrong Fragment do dataset NSL KDD.

```
static __always_inline __u8 getWrong_fragment( __be16 frag_off,
   __be16 tot_len )
{
  __u32 key =0;
  struct map_auxiliar *aux = bpf_map_lookup_elem(&xdp_auxiliar,&key )
     ;
    if (!aux)
      return XDP_ABORTED;
  //aux->wrong_fragment_calc = frag_off / (( tot_len-20)/8);
  aux->frag_off = frag_off;
        aux->tot_len = tot_len-20;
  aux->tot_len = aux->tot_len /8;
  aux->wrong_fragment_calc = aux->frag_off /aux->tot_len ;
  if(aux->wrong_fragment_calc == 0)
    return 0;
  else
    return 33;
}
```

Fonte: Autoria própria.

O próximo cabeçalho é analisado desde que retorne os três tipos suportados (TCP, UDP e ICMP). Desse modo, são extraídos os atributos *flag*, *land* e serviço, porém existem diferenças na extração desses atributos de acordo com o tipo de cabeçalho, tais particularidades serão descritas abaixo.

O atributo *flag* indica o estado atual da conexão. Durante a análise de pacotes UDP e ICMP o campo *flag* retorna SF, enquanto em conexões TCP o dataset possibilita 10 estados diferentes de conexão exibidos na Tabela 6, indicando se a conexão está aberta, foi encerrada normalmente ou não e, ainda, se aguarda alguma resposta da origem ou destino. Na lógica do sistema, é verificada se a conexão está aberta para que sejam atualizados os estados conforme a conexão prossegue. Em sequência, o atributo *land* implementado conforme o código fonte na Figura 15 verifica se o serviço de origem ou destino são os mesmos. Em pacotes do tipo

TCP, os campos tcp source e dest são comparados para verificar se a porta de origem e destino é a mesma e, nesse caso, *land* é setado como 1, ao contrário, esse valor é definido como 0 do mesmo modo que pacotes do tipo ICMP ou UDP.

Tabela 6 – Listagem das 10 possibilidades de estado em uma conexão TCP.

| **Nome** | **Descrição** |
|---|---|
| OTH 0 | Conexão sem SYN visto, apenas tráfego midstream (parte de uma conexão não foi fechada anteriormente). |
| REJ 10 | Tentativa de conexão rejeitada |
| RSTO 20 | Conexão estabelecida que é abortada pela origem através do envio de um RST. |
| RSTOS0 30 | Origem envia um SYN seguido de RST, entretanto, não recebe um syn-ack do destino. |
| RSTR 40 | Estabelecida, abortada pelo destino |
| S0 50 | Tentativa de conexão sem resposta |
| S1 60 | Conexão estabelecida não terminada. |
| S2 70 | Conexão foi estabelecida e ocorre uma tentativa de encerramento pela origem sem resposta do destino. |
| S3 80 | Conexão estabelecida e fechada pelo respondedor, sem resposta para origem. |
| SF 90 | Conexão normal estabelecida e terminada. |
| SH 100 | Origem envia um SYN seguido por um FIN, sem nunca ocorrer um SYN-ACK do destino. |

Fonte: Adaptado de (SONG; TAKAKURA; OKABE, 2006).

Figura 15 – Código fonte da extração do atributo Land do NSL KDD.

```
1  static __always_inline __u8 feature_land( struct tcphdr *tcp, int
       protocol )
2  {
3    if (protocol == IPPROTO_TCP) {
4      /* Caso a porta de servico seja a mesma */
5      if( tcp->source == tcp->dest  )
6        return 1;
7      else
8        return 0;
9    } else if (protocol == IPPROTO_UDP) {
10     return 0;
11   }
12   return 0;
13 }
```

Fonte: Autoria própria.

O atributo serviço é extraído do campo tcphdr->dest, que representa a porta de saída em conexões UDP e TCP, para tais conexões são definidas constantes com o nome do serviço, além de uma chave para comparação com o modelo. Na leitura dos pacotes UDP, é criado o *translator*, com o código disponível na Figura 16, que faz a tradução da porta do serviço disponível na conexão com o código do serviço gravado no modelo, essa conversão é baseada

no documento "*Service Name and Transport Protocol Port Number Registry*"do IANA (IANA, 2023). O processo de extração do atributo em conexões TCP é o mesmo, diferenciado apenas o número dos serviços contidos em cada protocolo. Em pacotes do tipo ICMP é necessária a análise nos campos *type* e code para chegar ao serviço disponível no modelo. Para isso, o código na Figura 17 funciona de maneira semelhante, entretanto, aqui é considerado o *type* do tipo 3 que retorna os serviços SRV_URP_I, SRV_URH_I e SR_OTH_I. E caso o *type* seja diferente de 3 são retornados os serviços listados nas condições apresentadas entre as linhas 12 e 27.

Figura 16 – Código fonte da extração do atributo Service UDP do NSL KDD.

```
...
#define SRV_OTHER  63 //799 //ok
#define SRV_DOMAIN_U 17 //53 // ok
#define SRV_TFTP_U 88  //#69 //ok
#define SRV_NTP_U 62 //123 //ko
...
static __always_inline __u8  udpTranslator(protocol, port){
  if( port >= 49152 )
    return SRV_PRIVATE;
  switch (service)
  {
     case 53:
     return SRV_DOMAIN_U;
     break;
     case 69:
     return SRV_TFTP_U;
     break;
     case 123:
     return SRV_NTP_U;
     break;
     default
     return SRV_OTHER;
  }
}
```

Fonte: Autoria própria.

Caso a leitura de todos os cabeçalhos seja concluída com sucesso, os atributos extraídos são gravados no mapa eBPF amostra (nslkdd_amostra). Sendo assim, ocorre a chamada do classificador KNN para categorizar os pacotes seguindo a lógica apresentada no Algoritmo 1. Desse modo, as partes principais do código fonte do classificador KNN eBPF serão apresentadas a seguir, divididas em 4 tarefas. Inicialmente, no código presente na Figura 19, é realizada a leitura da *struct* amostra, além da inicialização do mapa eBPF auxiliar (xdp_auxiliar) utilizado por todas as operações lógicas e matemáticas do programa. A segunda tarefa está disponível

Figura 17 – Código fonte da extração do atributo ICMP service do NSL KDD.

```
static __always_inline __u8  icmpTranlator(protocol, tipo, code){
    if ( tipo == 3 ){
        if( code == 0 )
            return SRV_URP_I;
        else
    if( code == 1 )
            return SRV_URH_I;
        else:
            return SRV_OTH_I;
    }
  switch (tipo)
  {
     case 0:
     return SRV_ECR_I;
     break;
     case 8:
     return SRV_ECO_I;
     break;
     case 5:
     return SRV_RED_I;
     break;
     case 11:
     return SRV_TIM_I;
     break;
     default
     return SRV_OTH_I;
  }
}
```

Fonte: Autoria própria.

no código na Figura 20, no qual é inicializado um laço de repetição que tem como tamanho o número de registros gravados no mapa eBPF modelo (nslkdd_mapa_selecionado). Durante essa fase, a aplicação realiza a leitura do modelo gravado, além de medir a distância Manhattan dos atributos escolhidos, usando a função eBPF disponível no código mostrado na Figura 18.

A terceira tarefa é exibida na Figura 21, nela são computados os vizinhos mais próximos através da comparação da amostra provinda do pacote com as gravadas no mapa eBPF. Essas amostras contém as classe 0 e 1, indicando um padrão de comportamento ataque ou normal. O código apresenta a definição de uma variável que armazena o vizinho mais próximo nomeado como K0, além da variável KClasse0 utilizada para guardar a classificação de K0. Esse código deve ser repetido para N vizinhos atualizando os índices do Ks (0,1,2) e também das classes (0,1,2), verificando se a distância computada entre a amostra do pacote é menor que a amostra do modelo. Nesse caso, o enésimo K e classe recebem os valores atuais, caso o K atual já

Figura 18 – Função eBPF de cálculo da distância Manhattan.

```
static __always_inline __u64 manhattan(__u64 a, __u64 b){
  __u32 key =0;
  struct map_auxiliar_distancia *aux2 = bpf_map_lookup_elem(&
     xdp_auxiliar_distancia,&key );
  if (!aux2)
    return XDP_ABORTED;
  aux2->a=a ;
  aux2->b =b;
  aux2->retorno =0;
  if(aux2->a < aux2->b){ ////// 4    -  6
    aux2->retorno =aux2->b-aux2->a;
    }
  else
  if( aux2->a== 0 && aux2->b > 0){
    aux2->retorno=aux2->b;
  }
  else
  if(aux2->b == 0 &&  aux2->a > 0)
  {
    aux2->retorno=aux2->a;
  }
  else{
    aux2->retorno=  aux2->a-aux2->b;
  }
  return aux2->retorno;
}
```

Fonte: Autoria própria.

tenha sido inicializado anteriormente, ao contrário apenas o K é definido. Portanto, é possível atualizar todos os valores dos vizinhos e suas respectivas classes baseadas nas amostras mais aproximadas do modelo. Por fim, na quarta etapa, exibida na Figura 21, as classe com as menores distâncias são somadas e a classificação é feita por votação majoritária.

Figura 19 – Primeira etapa do classificador.

```
static __always_inline__u32 xdp_classificador( ) {//1
    __u32 key2=0;
  struct knn_amostra  *rec2 = pf_map_lookup_elem(&nslkdd_amostra, &
     key2);
    if (!rec2)
      return XDP_ABORTED;
  __u8 protocol_type=rec2->protocol_type;//1
  __u8 service=rec2->service; //2
...
  struct map_auxiliar *aux = bpf_map_lookup_elem(&xdp_auxiliar,&key )
     ;
    if (!aux)
      return XDP_ABORTED;
...//2
```

Fonte: Autoria própria.

Figura 20 – Segunda etapa do classificador.

```
        aux->k0 = aux->k1 =aux->k3 = 101
  for(__u32 i = 0;i <size;i++){
    __u32 key2 = aux->i;
        struct  knn_features_selecionadas  *rec  =
           bpf_map_lookup_elem(&nslkdd_mapa_selecionado, &key2  );

        if (!rec)
          return XDP_ABORTED;
        aux->distancia=manhattan(rec->protocol_type,protocol_type);
        aux->distancia+=manhattan(rec->service,service);
      amostra = aux->distancia;
```

Fonte: Autoria própria.

Figura 21 – Terceira etapa do classificador.

```
      if(( aux->k0 > aux->distancia))
      {
                    if( aux->k1 > aux->k0){
                        aux->k2=aux->k1;
                        aux->kClasse2=aux->kClasse1;

                        aux->k1=aux->k0;
                        aux->kClasse1=aux->kClasse0;

                    }

        aux->k0 = aux->distancia;
        aux->kClasse0 =rec->classe ;
      }
      else
```

Fonte: Autoria própria.

Figura 22 – Quarta etapa do classificador.

```
...
   if(aux->kClasse0 == 1) /// normal 1 conforme funcao sklearn de
      conversao texto para numero
    aux->returnNormal++;
  else
    aux->returnAttack++;
  ...
  if(aux->returnNormal > aux->returnAttack )
  {
    aux->normal++;
    return 1;  /// normal 1 conforme funcao sklearn de conversao
       texto para numero
  }
  else
  {
    aux->ataque++;
    return 0; /// ataque 0 conforme funcao sklearn de conversao texto
        para numero
  }
...
}
```

Fonte: Autoria própria.

# 6 VALIDAÇÃO

Com o objetivo de validar a solução proposta, este Capítulo apresenta a experimentação prática. Os experimentos realizados foram conduzidos em duas etapas, com a primeira delas executada a partir de um conjunto de dados públicos contendo intrusões simuladas. Esse mesmo conjunto é oriundo de arquivos PCAPs, utilizados no segundo experimento, realizado em máquinas Virtuais (VMs) avaliadas ao limite em termos de menor quantidade de recursos possíveis, além da capacidade máxima de processamento.

## 6.1 METODOLOGIA

### 6.1.1 NSL-KDD Data set

O conjunto de dados KDDKUP 99 é amplamente utilizado na avaliação de inúmeros métodos de detecção, sendo originalmente criado para ser usado na "*Third Data Mining and Knowledge Discovery Competition on computer network intrusion detection*", que foi uma competição em que os competidores deveriam desenvolver um sistema capaz de auditar conexões realizadas. Essa base de dados é um subconjunto dos dados de 1998 coletado pelo DARPA, em uma simulação de operação da rede da Força Aérea dos Estados Unidos em uma rede local (LAN) (PROTIĆ, 2018). Os dados do DARPA 98 são cerca de 4 gigabytes de tráfego de rede em arquivos tcpdump, organizados em dias da semana entre segunda e sexta-feira, durante 7 semanas na etapa de treino, totalizando 5 milhões de conexões, além de conter duas semanas de dados de testes no mesmo formato com o total de 2 milhões de conexões.

Desse modo, o KDDCUP é um conjunto de dados gerados a partir de compilações dos dados de tráfego real do DARPA'98 com um total de 4,898,431 amostras de treino, além de 311,027 amostras de testes, em que cada uma delas representa uma conexão de rede em um comportamento normal ou então algum tipo de ataque. Os ataques contidos são do tipo DDoS, U2R, R2L e PROBE, totalizando 24 ataques nos dados de treino e 14 ataques adicionais nos dados de teste. Um dos objetivos da base de dados é fornecer aos pesquisadores meios para testar sistemas, principalmente para novos ataques ao invés de ficarem criando novas assinaturas, por esse motivo foram introduzidos os novos ataques (TAVALLAEE et al., 2009).

Entretanto, o KDDCUP foi duramente criticado por conter mais de 75% dos registros duplicados, o que pode levar detectores a resultados enviesados, por este motivo criou-se o

conjunto de dados NSL-KDD (TAVALLAEE et al., 2009), que tenta resolver os problemas de registros redundantes, seleciona também um número de registros em quantidades razoáveis disponíveis nas bases de dados de treino e teste, além de distribuir proporcionalmente o número de registros do conjunto de dados original.

Os testes realizados nos experimentos deste trabalho ocorreram em duas fases; a primeira delas, realizada off-line, é referente a elaboração e avaliação do conjunto de dados NSL-KDD em diferentes configurações a fim de verificar a sanidade do algoritmo utilizado no classificador desenvolvido e também a seleção de uma configuração adequada ao mesmo. Nos experimentos realizados off-line, utilizou-se a linguagem de programação Python, com a biblioteca Scikit Learning como algoritmo de ML popular, sendo este comparado a um algoritmo próprio que também foi utilizado na escolha do conjunto de atributos empregado pelo modelo gerado. Durante a segunda parte dos experimentos, é utilizada uma seleção de dados extraídos dos arquivos PCAPs do KDDCUP para verificação online do funcionamento do detector criado. Essa seleção forma 2 novos arquivos PCAPs com 40 segundos de tráfego de rede. Em um dos arquivos existe apenas tráfego normal, no outro, existe também os ataques selecionados. Cada um desses arquivos é enviado com as opções do TPCREPLAY de duração de 40 segundos e, também, para acelerar o envio dos pacotes na intenção de estagnar a interface de rede na sua máxima capacidade. Os dois arquivos PCAPs são capturados individualmente pelo software Wireshark, formando os arquivos usados no experimento, que tem uma duração de aproximadamente 80 segundos.

### 6.1.2 Métricas usadas na validação do experimento

Um detector de ataques baseado em eBPF pode trazer benefícios a infraestrutura de rede, garantindo baixo requisito de hardware, segurança, flexibilidade, maior desempenho e redução de custos com equipamentos devido à flexibilidade imposta pela programabilidade. A avaliação de seu desempenho nos termos de identificação de ataques, além de requisitos de hardware, é analisada com as métricas definidas em (QUINCOZES et al., 2020), além da vazão que é utilizada na aferição do tráfego enviado:

- Verdadeiro Positivo (VP) - Conexão anômala corretamente classificada.
- Verdadeiro Negativo (VN) - Conexão normal corretamente classificada.
- Falso Positivo (FP) - Conexão Normal classificada erroneamente como anomalia.

- Falso Negativo (FN) - Conexão anômala classificada erroneamente como conexão normal
- *Recall* (Sensibilidade) - Relação de detecções Positivas classificadas corretamente (Equação 6.1).
- Precision (Especificidade) - Relação de detecções de tráfego normal classificados corretamente (Equação 6.2).
- *F1-Score* - É uma medida geral de precisão baseada na combinação de Sensibilidade e Especificidade, que verifica a capacidade de identificar falsos positivos e falsos negativos (Equação 6.4).
- Acurácia - Verifica a capacidade de predições corretas sobre o total de predições (Equação 6.3).
- Vazão - Representa a maior taxa possível de encaminhamento de pacotes em determinado período de tempo por um dispositivo, contabilizada em Mega bytes por segundo.

$$Recall = \frac{TP}{TP + FN} \tag{6.1}$$

$$Precision = \frac{TP}{TP + FP} \tag{6.2}$$

$$Acurcia = \frac{TP + TN}{TP + FP + TN + FN} \tag{6.3}$$

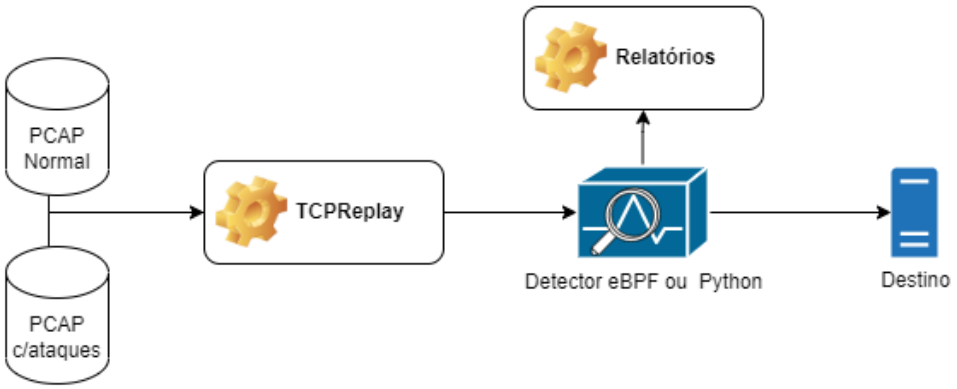
$$F1 - Score = 2 * \frac{Precision * Recall}{PrecisionxRecall} \tag{6.4}$$

### 6.1.3 Ambiente de testes

O ambiente de testes é realizado em ambiente virtualizado em um computador com processador AMD FX 8350 4.0 GHz de 8 núcleos, memória de 12 Gigabytes. Os testes são executados em máquina virtual com configurações variadas, onde se buscou configurar no mínimo aceitável para inicializar o sistema. Os testes são realizados através do envio simultâneo dos dois arquivos PCAPs gerados, de maneira que a ferramenta TCP replay emite um relatório do envio dos pacotes de ambos os arquivos e os envia. Tcpreplay é um utilitário *Open Source*

para edição e reprodução de tráfego de rede previamente capturado, originalmente foi projetado para reproduzir padrões de tráfego malicioso para Sistemas de Detecção/Prevenção de intrusos. Em sua operação básica ele pode reenviar todos os pacotes de um arquivo PCAP, por exemplo, na velocidade em que ele foi gravado. Entretanto, é possível ajustar alguns parâmetros para que o tráfego seja reenviado na velocidade que o hardware é capaz de enviar. Além do relatório da ferramenta, os classificadores eBPF e Python gravam os resultados das classificações realizadas. Esse ambiente de simulação é retratado na Figura 23.

Figura 23 – Ambiente de testes realizados em máquina Virtual.




Fonte: Autoria própria.

## 6.2 VALIDAÇÃO EXPERIMENTAL

Nesta seção são apresentados os resultados obtidos a partir dos experimentos realizados, nos quais foram comparados os resultados do algoritmo proposto com os de um algoritmo de uma biblioteca popular de programação. Também são mostrados resultados do experimento de tempo de envio e largura de banda, que são obtidos através dos testes realizados com tráfego de rede utilizando arquivos '.pcap', dos quais foram selecionadas amostras de um ataque e tráfego normal e, por fim, é demonstrado um experimento para verificar se a quantidade de atributos tem impacto no tempo de processamento.

### 6.2.1 Experimento de sanidade do algoritmo simplificado

A fim de validar o algoritmo KNN proposto, a Equação 6.5 é utilizada na verificação de todas as combinações possíveis ao conjunto de 5 atributos, sendo que os atributos selecionados na etapa de seleção demonstrados na Tabela 3 são 'protocol_type', 'service', 'flag', 'land',

'wrong_fragment'. No entanto, com vistas a legitimar o funcionamento deste algoritmo, ele foi comparado ao de uma biblioteca popular. Os resultados visíveis na Tabela 7 apresentam alguma divergência, no entanto, são semelhantes em ambos algoritmos. São mostrados 12 resultados do algoritmo KNN disponível na biblioteca Scikit-Learn e 12 resultados do L-KNN (Os dez melhores e os dois piores resultados), dessa maneira é possível verificar que o algoritmo utilizado no classificador de Kernel e Espaço do usuário tem um desempenho aproximado a essa biblioteca. Outro ponto a destacar é referente aos resultados do L-KNN serem próximos em diversos conjuntos de atributos, isso ocorre devido a maneira na qual a distância foi calculada levando em consideração os números de 0 até 100. No entanto é necessário que ocorram mais testes para verificar como resolver essas questões.

$$C_{n,p} = \frac{n!}{p!(n-p)!} \tag{6.5}$$

Tabela 7 – Comparação de resultados nos algoritmos L-KNN e Scikit-learn.

| **Light KNN** | | **Python Scikit learn** | |
|---|---|---|---|
| **Features utilizada(s)** | **Accuracy** | **Features utilizada(s)** | **Accuracy** |
| **service, flag** | 79,12% | **service,flag** | 78,57% |
| **protocol_type, service, flag** | 79,12% | **protocol_type,service,flag** | 79,22% |
| **service, flag, land** | 79,12% | **service,flag,land** | 78,94% |
| **service, flag, wrong_fragment** | 79,12% | **service,flag,wrong_fragment** | 79,92% |
| **protocol_type, service, flag, land** | 79,12% | **protocol_type,service,flag,land** | 79,13% |
| **protocol_type, service, flag, wrong_fragment** | 79,12% | **protocol_type,service,flag,wrong_fragment** | 79,11% |
| **service, flag, land, wrong_fragment** | 79,12% | **service,flag,land,wrong_fragment** | 79,89% |
| **protocol_type, service, flag, land, wrong_fragment** | 79,12% | **protocol_type,service,flag,land,wrong_fragment** | 77,74% |
| **protocol_type, flag** | 78,91% | **protocol_type,flag** | 78,91% |
| **protocol_type, flag, land** | 78,91% | **protocol_type,flag,land** | 78,88% |
| **wrong_fragment** | 43,06% | **'wrong_fragment** | 43,06% |
| **land** | 43,08% | **land** | 43,11% |

Fonte: Autoria própria.

### 6.2.2 Tempo de processamento e pacotes enviados por segundo

No experimento realizado são gerados dois arquivos PCAPs com cerca de quarenta segundos de tráfego cada um deles, os quais contêm o total de 172111 pacotes normais e ataques, com o número de pacotes pertencendo as classes normal e ataque em respectivamente 37547 e 134564 amostras. A ferramenta TCPReplay é utilizada no envio de ambos PCAPS, possibilitando computar o tempo de envio e também a largura de banda em experimentos realizados com

diferentes configurações de memória, processador e, também, o classificador eBPF no Kernel do sistema e, ainda, o classificador desenvolvido com a linguagem Python. Uma vez selecionado o tipo de ataque no experimento é gerada uma base de dados com este tipo de ataque e dados normais, o qual é avaliado pelo módulo de seleção de atributos que indica os atributos mais relevantes, que nesse caso foram protocolo e serviço. Com isso é gerado um modelo utilizado pelos classificadores conforme descrito no Capítulo 4.

Na avaliação do tempo de envio e processamento desses pacotes são utilizadas as configurações na aplicação de Kernel sem classificador e um classificador com filtro eBPF e, também, da avaliação no espaço do usuário do classificador Python. O experimento foi realizado em Máquinas Virtuais (VMs) com avaliações de diferentes classificadores em que a memória RAM é alterada entre 256MB até 8196MB, além de também ser alterada a configuração na quantidade de processadores em cada experimento realizado. O valor de 256MB foi escolhido devido a impossibilidade da VM inicializar com menos recursos.

O primeiro teste foi realizado com apenas 512 MB de memória e 1 processador. Em função do tempo de processamento ser elevado com o uso de apenas um processador, decidiu-se realizar o restante do experimento com 2 processadores, visto que, nessa configuração, é possível obter tempos melhores mesmo com configurações de memória RAM baixas. Os experimentos realizados aconteceram do seguinte modo: com 1 processador e 512MB, 2 processadores com 256MB, 512MB, 4096MB e 8192 MB e, por fim, com 4 processadores e 8192MB. Para essas configurações foram executados 2 testes para aferição do tempo de processamento e pacotes enviados por segundos. As comparações dos resultados podem ser visualizadas na Tabela 8. É possível acrescentar que os testes feitos com memórias mais baixas foram realizados para validar a utilização de configurações reduzidas. Em contrapartida, as configurações de 4GB e 8GB foram uma tentativa de simular a capacidade de switches fabricados em 2021 e 2022 que partem de 4GB de memória, sendo possível verificar a informação no datasheet do Switch Dell (DELL, 2022) e Cisco (CISCO SYSTEMS, 2022).

Além da medição dos tempos de envio e pacotes enviados por segundo, foi avaliada a capacidade do classificador em detectar ataques. Desse modo, o classificador eBPF é implantado no driver do dispositivo de rede, executando o algoritmo L-KNN obteve o *F1-Score* e acurácia de 98%, *Recall* 100% e precisão de 95% para o tipo de ataque DDoS. No resultado, o tráfego de rede obtido dos arquivos PCAPs é analisado.

Tabela 8 – Tempo de processamento e vazão para os cenários avaliados.

| Cenário | Configuração | | | Tempo de Processamento | Pacotes por Segundos |
|---|---|---|---|---|---|
| **Rank** | **Memória** | **Processadores** | **Classificador** | **Segundos (s)** | **Megabytes por Segundo (Mbps)** |
| **1** | 256 | 2 | Sem classificador | 81,52 | 28,81 |
| **2** | 8192 | 4 | Sem classificador | 81,52 | 28,81 |
| **3** | 512 | 2 | Sem classificador | 81,54 | 28,79 |
| **4** | 4096 | 2 | Sem classificador | 81,54 | 28,81 |
| **7** | 8192 | 2 | Sem classificador | 81,62 | 28,77 |
| **11** | 512 | 1 | Sem classificador | 82,07 | 28,53 |
| **5** | 4096 | 2 | KNN-eBPF | 81,57 | 28,81 |
| **6** | 512 | 2 | KNN-eBPF | 81,61 | 28,76 |
| **8** | 8192 | 2 | KNN-eBPF | 81,63 | 28,78 |
| **9** | 8192 | 4 | KNN-eBPF | 81,63 | 28,79 |
| **10** | 256 | 2 | KNN-eBPF | 81,78 | 28,69 |
| **12** | 512 | 1 | KNN-eBPF | 82,31 | 28,4 |
| **13** | 8192 | 4 | KNN-Python | 87,11 | 25,94 |

Fonte: Autoria própria.

Tabela 9 – Resumo de diferença em percentual entre configurações avaliadas na Tabela, em que o ID representa os melhores resultados de cada configuração 8.

| ID | Tempo de processamento | MB enviados por segundos Mbps |
|---|---|---|
| 1 | 100% | 100% |
| 5 | 99,94% | 100% |
| 13 | 93,58% | 90,04% |

Fonte: Autoria própria.

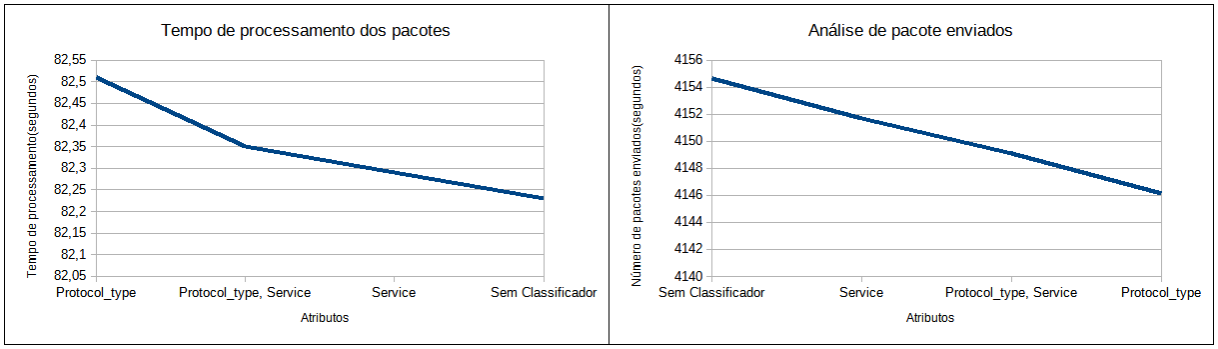
### 6.2.3 Impacto de Atributos

Com a finalidade de aferir se existe algum impacto de desempenho em relação a quantidade de atributos utilizados, conduziu-se mais um experimento comparando o desempenho do equipamento em transmitir os dados sem classificador e com o serviço de classificação. No experimento realizado através do modo de seleção foram selecionados os atributos protocolo e serviço, onde se realizou medições de tempo de envio e pacotes enviados por segundos. Para isso se avaliou uma configuração sem classificador, classificação com cada um dos dois atributos avaliados individualmente e, por fim, avaliados os dois atributos.

Os resultados mostram uma ligeira variação em relação ao tipo e quantidade de atributos utilizados, de maneira que, novamente, o tempo de envio e a vazão são ligeiramente melhores sem o uso do classificador e, a medida que acrescentamos atributos ou alteramos atributos, a vazão decai superficialmente. Numa rápida análise dos valores desse experimento é possível

verificar que a melhor vazão é Sem classificador, mas com a utilização do serviço de classificação eBPF não existe um prejuízo expressivo de vazão. Para essa averiguação, os resultados podem ser visualizados na Figura 24.

Figura 24 – Comparação do impacto do classificador considerando o tipo ou quantidade de atributos, no processamento de pacotes.




Fonte: Autoria própria.

## 6.3 DISCUSSÃO

Após a elaboração dos experimentos, os resultados demonstram que a aplicação do kernel tem capacidade para lidar com altas demandas de tráfego de rede e processamento de pacotes. Em adicional, também requer uma baixa quantidade de recursos computacionais, o que não causa impactos significativos em experimentos realizados, considerando a máxima capacidade de processamento permitido pelo hardware utilizado. Portanto, os resultados demonstram que a proposta pode ser útil dentro de estruturas de redes nos equipamentos presentes nelas. Ademais, esta aplicação pode ser utilizada em ambiente de rede e no reconhecimento de ataques realizados por dispositivos internos ou externos uma vez que pode ser posicionada em switches existentes que ficam próximos aos dispositivos finais utilizados pelos usuários. O algoritmo KNN, por sua vez, pode ter relevância em alguns cenários de ataques na identificação destas ameaças. Entretanto, durante a elaboração do experimento, houve a percepção de uma tendência de que sua utilização seja melhor aplicável em ambiente com baixo número de ataques, gerando um mapa eBPF mais compatível com o hardware existente. Devido às restrições de memória averiguadas pelo verificador JIT, muitos ataques gerariam um mapa grande, ocasionando a rejeição do programa se o seu tamanho exceder o limite permitido pelo verificador. Ainda assim, durante o experimento, é possível detectar ataques mesmo quando a interface de rede está operando em sua capacidade máxima, sem afetar o desempenho dessa interface.

# 7 CONSIDERAÇÕES FINAIS

Tendo em vista os aspectos observados no decorrer do trabalho, é possível discorrer acerca da percepção de alguns benefícios na utilização de um classificador eBPF, considerando-se que pode ser implantado ao nível do Kernel na análise de pacotes diretamente no *driver* do dispositivo de rede. O Kernel evita que o processamento seja realizado nas camadas mais altas do espaço do usuário. Com isso, o trabalho em questão mostra as etapas de desenvolvimento e implantação de um classificador L-KNN que pode ser colocado em dispositivos de rede como interfaces de switches ou então placas de rede compatíveis com eBPF. Os testes apresentados mostram benefícios na utilização do classificador eBPF em termos de segurança, custo e também desempenho. Entretanto, durante a implementação e testes, fica evidenciado que o algoritmo KNN não é a melhor opção para um classificador nessa modalidade de dispositivo, devido à escalabilidade do MAPA eBPF, gerado. Conforme aumenta a quantidade de atributos ou, amostras analisadas, há impacto direto no tamanho do programa, o qual é limitado. Há também a necessidade de implementar um algoritmo de busca que auxilia na redução de comparações realizadas pelo algoritmo KNN, que tem como ponto positivo o fato de ser amplamente utilizado na literatura, pois obtém resultados consistentes, de fácil implementação e não havia sido implementado nesse tipo de equipamento. Do mesmo modo, o KNN também pode ser facilmente empregado para aumentar o tamanho do programa eBPF e avaliar o desempenho da análise do tráfego de rede com programas grandes.

Em trabalhos futuros, pode ser realizada a implementação do armazenamento de estados e estatísticas, além do uso de algoritmos como árvore de decisão devido a facilidade de emprego com eBPF, além de Rede Neural, dificultado pela limitação existente em operações matemáticas e, com isso, deve-se optar por trabalhar com operações simples ou desenvolver funções mais complexas de cálculos ao custo de desempenho na classificação. Também pode ser realizado um estudo com diferentes ataques, onde o componente treinador ajusta equipamentos para reconhecerem as atividades de seus usuários e dos atacantes. Outro ponto que precisa de investigação é referente ao tratamento de anomalias identificadas pelo sistema proposto, sendo necessário definir a ação tomada pela aplicação.

# REFERÊNCIAS

ALSABEH, A. et al. A survey on security applications of p4 programmable switches and a stride-based vulnerability assessment. **Computer Networks**, [S.l.], v.207, p.108800, 2022.

AMROLLAHI, M. et al. Enhancing network security via machine learning: opportunities and challenges. **Handbook of big data privacy**, [S.l.], p.165–189, 2020.

BACHL, M.; FABINI, J.; ZSEBY, T. A flow-based IDS using Machine Learning in eBPF. **arXiv preprint arXiv:2102.09980**, [S.l.], 2021.

BARRADAS, D. et al. FlowLens: enabling efficient flow classification for ml-based network security applications. In: NDSS. **Anais...** [S.l.: s.n.], 2021.

BEN-YAIR, I.; ROGOVOY, P.; ZAIDENBERG, N. AI & eBPF based performance anomaly detection system. In: ACM INTERNATIONAL CONFERENCE ON SYSTEMS AND STORAGE, 12. **Proceedings...** [S.l.: s.n.], 2019. p.180–180.

BERTIN, G. XDP in practice: integrating xdp into our ddos mitigation pipeline. In: TECHNICAL CONFERENCE ON LINUX NETWORKING, NETDEV. **Anais...** [S.l.: s.n.], 2017. v.2, p.1–5.

BIFULCO, R.; RÉTVÁRI, G. A survey on the programmable data plane: abstractions, architectures, and open problems. In: IEEE 19TH INTERNATIONAL CONFERENCE ON HIGH PERFORMANCE SWITCHING AND ROUTING (HPSR), 2018. **Anais...** [S.l.: s.n.], 2018. p.1–7.

CARVALHO BERTOLI, G. de et al. Evaluation of netfilter and eBPF/XDP to filter TCP flag-based probing attacks. In: XXII SYMPOSIUM ON OPERATIONAL APPLICATIONS IN DEFENSE AREA (SIGE). **Proceedings...** [S.l.: s.n.], 2020. p.35–39.

CARVALHO, D. et al. BG-IDPS: detecção e prevenção de intrusões em tempo real em switches ebpf com o filtro de pacotes berkeley e a metaheurística grasp-fs. In: XXII SIMPÓSIO BRASILEIRO EM SEGURANÇA DA INFORMAÇÃO E DE SISTEMAS COMPUTACIONAIS. **Anais...** [S.l.: s.n.], 2022. p.139–152.

CEROVIĆ, D. et al. Fast packet processing: a survey. **IEEE Communications Surveys & Tutorials**, [S.l.], v.20, n.4, p.3645–3676, 2018.

CISCO SYSTEMS, I. **Cisco Catalyst 9300 Series Switches Data Sheet**. Acessado: em Junho/2022, https://www.cisco.com/c/en/us/products/collateral/switches/catalyst-9300-series-switches/nb-06-cat9300-ser-data-sheet-cte-en.html.

CORMODE, G.; MUTHUKRISHNAN, S. An improved data stream summary: the count-min sketch and its applications. **Journal of Algorithms**, [S.l.], v.55, n.1, p.58–75, 2005.

DELL. **Switches do Dell EMC PowerSwitch Série N3200-ON**. Acessado: em Junho/2022, https://www.delltechnologies.com/asset/pt-br/products/networking/technical-support/dell-networking-n3200-powerswitch-specsheet.pdf.

DEPREN, O. et al. An intelligent intrusion detection system for anomaly and misuse detection in computer networks. **Expert systems with Applications**, [S.l.], v.29, n.4, p.713–722, 2005.

DHANABAL, L.; SHANTHARAJAH, S. A study on NSL-KDD dataset for intrusion detection system based on classification algorithms. **International journal of advanced research in computer and communication engineering**, [S.l.], v.4, n.6, p.446–452, 2015.

DIMOLIANIS, M.; PAVLIDIS, A.; MAGLARIS, V. A multi-feature DDoS detection schema on P4 network hardware. In: CONFERENCE ON INNOVATION IN CLOUDS, INTERNET AND NETWORKS AND WORKSHOPS (ICIN), 2020. **Anais...** [S.l.: s.n.], 2020. p.1–6.

DIMOLIANIS, M.; PAVLIDIS, A.; MAGLARIS, V. Signature-Based Traffic Classification and Mitigation for DDoS Attacks Using Programmable Network Data Planes. **IEEE Access**, [S.l.], v.9, p.113061–113076, 2021.

DING, D. et al. In-network volumetric DDoS victim identification using programmable commodity switches. **IEEE Transactions on Network and Service Management**, [S.l.], v.18, n.2, p.1191–1202, 2021.

DING, D.; SAVI, M.; SIRACUSA, D. Tracking normalized network traffic entropy to detect DDoS attacks in P4. **IEEE Transactions on Dependable and Secure Computing**, [S.l.], v.19, n.6, p.4019–4031, 2021.

EBPF. **What is eBPF**. 2023.

ESTAN, C.; VARGHESE, G.; FISK, M. Bitmap algorithms for counting active flows on high speed links. In: ACM SIGCOMM CONFERENCE ON INTERNET MEASUREMENT, 3. **Proceedings...** [S.l.: s.n.], 2003. p.153–166.

FAYAZ, S. K. et al. Bohatei: flexible and elastic ddos defense. In: USENIX} SECURITY SYMPOSIUM ({USENIX} SECURITY 15), 24. **Anais...** [S.l.: s.n.], 2015. p.817–832.

FEO, T. A.; RESENDE, M. G. A probabilistic heuristic for a computationally difficult set covering problem. **Operations research letters**, [S.l.], v.8, n.2, p.67–71, 1989.

FERNANDES, G. et al. A comprehensive survey on network anomaly detection. **Telecommunication Systems**, [S.l.], v.70, p.447–489, 2019.

FOUNDATION eBPF. **What is eBPF? An Introduction and Deep Dive into the eBPF Technology**. Acessado: em julho/2023, https://ebpf.io/what-is-ebpf/hook-overview.

GAO, S.; HANDLEY, M.; VISSICCHIO, S. Stats 101 in p4: towards in-switch anomaly detection. In: ACM WORKSHOP ON HOT TOPICS IN NETWORKS. **Proceedings...** [S.l.: s.n.], 2021. p.84–90.

GAO, Y.; WANG, Z. A Review of P4 Programmable Data Planes for Network Security. **Mobile Information Systems**, [S.l.], v.2021, 2021.

GOLCHIN, P. et al. In-Network SYN Flooding DDoS Attack Detection Utilizing P4 Switches. , [S.l.], 2022.

GRAY, N. et al. High performance network metadata extraction using P4 for ML-based intrusion detection systems. In: IEEE 22ND INTERNATIONAL CONFERENCE ON HIGH PERFORMANCE SWITCHING AND ROUTING (HPSR), 2021. **Anais...** [S.l.: s.n.], 2021. p.1–7.

GUTIÉRREZ, S. A. et al. Watching Smartly from the Bottom: intrusion detection revamped through programmable networks and artificial intelligence. **arXiv preprint arXiv:2106.00239**, [S.l.], 2021.

HAUSER, F. et al. A survey on data plane programming with p4: fundamentals, advances, and applied research. **arXiv preprint arXiv:2101.10632**, [S.l.], 2021.

IANA. **Internet Assigned Numbers Authority**. Acessado: em Abril/2023, https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml.

KEHOE, M. {eBPF}: the next power tool of {SREs}. , [S.l.], 2022.

KFOURY, E. F.; CRICHIGNO, J.; BOU-HARB, E. An exhaustive survey on p4 programmable data plane switches: taxonomy, applications, challenges, and future trends. **IEEE Access**, [S.l.], v.9, p.87094–87155, 2021.

KHAMMASSI, C.; KRICHEN, S. A GA-LR wrapper approach for feature selection in network intrusion detection. **computers & security**, [S.l.], v.70, p.255–277, 2017.

KHOOI, X. Z. et al. DIDA: distributed in-network defense architecture against amplified reflection ddos attacks. In: IEEE CONFERENCE ON NETWORK SOFTWARIZATION (NETSOFT), 2020. **Anais...** [S.l.: s.n.], 2020. p.277–281.

KHRAISAT, A. et al. Survey of intrusion detection systems: techniques, datasets and challenges. **Cybersecurity**, [S.l.], v.2, n.1, p.1–22, 2019.

KUKA, M. et al. Accelerated DDoS Attacks Mitigation using Programmable Data Plane. In: ACM/IEEE SYMPOSIUM ON ARCHITECTURES FOR NETWORKING AND COMMUNICATIONS SYSTEMS (ANCS), 2019. **Anais...** [S.l.: s.n.], 2019. p.1–3.

KUKA, M. et al. Accelerated DDoS attacks mitigation using programmable data plane. In: ACM/IEEE SYMPOSIUM ON ARCHITECTURES FOR NETWORKING AND COMMUNICATIONS SYSTEMS (ANCS), 2019. **Anais...** [S.l.: s.n.], 2019. p.1–3.

LAPOLLI, Â. C.; MARQUES, J. A.; GASPARY, L. P. Offloading real-time DDoS attack detection to programmable data planes. In: IFIP/IEEE SYMPOSIUM ON INTEGRATED NETWORK AND SERVICE MANAGEMENT (IM), 2019. **Anais...** [S.l.: s.n.], 2019. p.19–27.

LEWIS, B.; BROADBENT, M.; RACE, N. P4ID: p4 enhanced intrusion detection. In: IEEE CONFERENCE ON NETWORK FUNCTION VIRTUALIZATION AND SOFTWARE DEFINED NETWORKS (NFV-SDN), 2019. **Anais...** [S.l.: s.n.], 2019. p.1–4.

LI, G. et al. Enabling performant, flexible and cost-efficient DDoS defense with programmable switches. **IEEE/ACM Transactions on Networking**, [S.l.], v.29, n.4, p.1509–1526, 2021.

LIN, W.-C.; KE, S.-W.; TSAI, C.-F. CANN: an intrusion detection system based on combining cluster centers and nearest neighbors. **Knowledge-based systems**, [S.l.], v.78, p.13–21, 2015.

LIU, Z. et al. Jaqen: a {High-Performance}{Switch-Native} approach for detecting and mitigating volumetric {DDoS} attacks with programmable switches. In: USENIX SECURITY SYMPOSIUM (USENIX SECURITY 21), 30. **Anais...** [S.l.: s.n.], 2021. p.3829–3846.

MAURIZIO, M. **Master Degree in Computer Engineering**. 2017. Tese (Doutorado em Ciência da Computação) — POLITECNICO DI TORINO.

MCKEOWN, N. et al. OpenFlow: enabling innovation in campus networks. **ACM SIGCOMM computer communication review**, [S.l.], v.38, n.2, p.69–74, 2008.

MIANO, S. et al. Creating complex network services with ebpf: experience and lessons learned. In: IEEE 19TH INTERNATIONAL CONFERENCE ON HIGH PERFORMANCE SWITCHING AND ROUTING (HPSR), 2018. **Anais...** [S.l.: s.n.], 2018. p.1–8.

MICHEL, O. et al. The Programmable Data Plane: abstractions, architectures, algorithms, and applications. **ACM Comput. Surv.**, New York, NY, USA, v.54, n.4, may 2021.

MUSUMECI, F. et al. Machine-learning-assisted DDoS attack detection with P4 language. In: ICC 2020-2020 IEEE INTERNATIONAL CONFERENCE ON COMMUNICATIONS (ICC). **Anais...** [S.l.: s.n.], 2020. p.1–6.

MUSUMECI, F. et al. Machine-learning-enabled DDoS attacks detection in P4 programmable networks. **Journal of Network and Systems Management**, [S.l.], v.30, p.1–27, 2022.

PRADHAN, P. S.; MANNEPALLI, P. K. Machine Leaning for Flow Based Intrusion Detection Using Extended Berkley Packet Filter. , [S.l.], 2021.

PROTIĆ, D. D. Review of KDD Cup '99, NSL-KDD and Kyoto 2006+ datasets. **Vojnotehnički glasnik/Military Technical Courier**, [S.l.], v.66, n.3, p.580–596, 2018.

QIN, Q. et al. Line-Speed and Scalable Intrusion Detection at the Network Edge via Federated Learning. In: IFIP NETWORKING CONFERENCE (NETWORKING), 2020. **Anais...** [S.l.: s.n.], 2020. p.352–360.

QIN, Q. et al. Line-speed and scalable intrusion detection at the network edge via federated learning. In: IFIP NETWORKING CONFERENCE (NETWORKING), 2020. **Anais...** [S.l.: s.n.], 2020. p.352–360.

QUINCOZES, S. E. et al. GRASP-based Feature Selection for Intrusion Detection in CPS Perception Layer. In: CONFERENCE ON CLOUD AND INTERNET OF THINGS (CIOT), 2020. **Anais...** [S.l.: s.n.], 2020. p.41–48.

QUINCOZES, S. E. et al. On the performance of GRASP-based feature selection for CPS intrusion detection. **IEEE Transactions on Network and Service Management**, [S.l.], v.19, n.1, p.614–626, 2021.

ROSHANI, M.; NOBAKHT, M. HybridDAD: detecting ddos flooding attack using machine learning with programmable switches. In: INTERNATIONAL CONFERENCE ON AVAILABILITY, RELIABILITY AND SECURITY, 17. **Proceedings...** [S.l.: s.n.], 2022. p.1–11.

SANTOYO-GONZÁLEZ, A.; CERVELLÓ-PASTOR, C.; PEZAROS, D. P. High-performance, platform-independent DDoS detection for IoT ecosystems. In: IEEE 44TH CONFERENCE ON LOCAL COMPUTER NETWORKS (LCN), 2019. **Anais...** [S.l.: s.n.], 2019. p.69–75.

SCIKIT-LEARN. **Repositories related to the scikit-learn Python machine learning library**. Acessado: em Junho/2022, https://github.com/scikit-learn/scikit-learn/blob/ada2f6bb20784e90072a6518e74f04513a7d448e/sklearn/neighbors/_classification.py#L611.

SILVEIRA ILHA, A. da et al. Euclid: a fully in-network, p4-based approach for real-time ddos attack detection and mitigation. **IEEE Transactions on Network and Service Management**, [S.l.], v.18, n.3, p.3121–3139, 2020.

SIMSEK, G. et al. Dropppp: a p4 approach to mitigating dos attacks in sdn. In: INFORMATION SECURITY APPLICATIONS: 20TH INTERNATIONAL CONFERENCE, WISA 2019, JEJU ISLAND, SOUTH KOREA, AUGUST 21–24, 2019, REVISED SELECTED PAPERS 20. **Anais...** [S.l.: s.n.], 2020. p.55–66.

SONG, J.; TAKAKURA, H.; OKABE, Y. Description of kyoto university benchmark data. **Available at link: http://www. takakura. com/Kyoto_data/BenchmarkData-Description-v5. pdf [Accessed on 15 March 2016]**, [S.l.], 2006.

SUN, C. et al. More accurate and fast SYN flood detection. In: PROCEEDINGS OF 18TH INTERNATIONAL CONFERENCE ON COMPUTER COMMUNICATIONS AND NETWORKS, 2009. **Anais...** [S.l.: s.n.], 2009. p.1–6.

TAVALLAEE, M. et al. A detailed analysis of the KDD CUP 99 data set. In: IEEE SYMPOSIUM ON COMPUTATIONAL INTELLIGENCE FOR SECURITY AND DEFENSE APPLICATIONS, 2009. **Anais...** [S.l.: s.n.], 2009. p.1–6.

TAVARES, K.; FERRETO, T. DDoS on Sketch: spoofed ddos attack defense with programmable data plans using sketches in sdn. In: XXXVII SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES E SISTEMAS DISTRIBUÍDOS. **Anais...** [S.l.: s.n.], 2019. p.805–819.

TU, W.; RUFFY, F.; BUDIU, M. Linux network programming with p4. In: LINUX PLUMB. CONF. **Anais...** [S.l.: s.n.], 2018.

UMAR, M. A.; ZHANFANG, C. Effects of Feature Selection and Normalization on Network Intrusion Detection. , [S.l.], 2020.

VIEIRA, M. A. et al. Processamento Rápido de Pacotes com eBPF e XDP. **Sociedade Brasileira de Computação**, [S.l.], 2019.

VIEIRA, M. A. et al. Fast packet processing with ebpf and xdp: concepts, code, challenges, and applications. **ACM Computing Surveys (CSUR)**, [S.l.], v.53, n.1, p.1–36, 2020.

VINAYAKUMAR, R. et al. Deep learning approach for intelligent intrusion detection system. **Ieee Access**, [S.l.], v.7, p.41525–41550, 2019.

WANG, S.-Y.; CHANG, J.-C. Design and implementation of an intrusion detection system by using Extended BPF in the Linux kernel. **Journal of Network and Computer Applications**, [S.l.], v.198, p.103283, 2022.

WANG, S.-Y.; CHANG, J.-C. Design and implementation of an intrusion detection system by using Extended BPF in the Linux kernel. **Journal of Network and Computer Applications**, [S.l.], v.198, p.103283, 2022.

XING, J. et al. A Vision for Runtime Programmable Networks. In: TWENTIETH ACM WORKSHOP ON HOT TOPICS IN NETWORKS, New York, NY, USA. **Proceedings...** Association for Computing Machinery, 2021. p.91–98. (HotNets '21).

ZHANG, M. et al. Poseidon: mitigating volumetric ddos attacks with programmable switches. In: NETWORK AND DISTRIBUTED SYSTEM SECURITY SYMPOSIUM (NDSS 2020), 27. **Anais...** [S.l.: s.n.], 2020.

ZHOU, Y. et al. Building an efficient intrusion detection system based on feature selection and ensemble classifier. **Computer Networks**, [S.l.], v.174, p.107247, 2020.