UNIVERSIDADE FEDERAL FLUMINENSE

HELIO DO NASCIMENTO CUNHA NETO

MineCap: Detecção de Mineração de Criptomoedas em Redes Corporativas com Aprendizado de Máquina e Prevenção de Abusos com Redes Definidas por Software

NITERÓI

2019

UNIVERSIDADE FEDERAL FLUMINENSE

HELIO DO NASCIMENTO CUNHA NETO

MineCap: Detecção de Mineração de Criptomoedas em Redes Corporativas com Aprendizado de Máquina e Prevenção de Abusos com Redes Definidas por Software

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e de Telecomunicações da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre em Engenharia Elétrica e de Telecomunicações. Área de concentração: Sistemas de Telecomunicações

Orientadora:
NATALIA CASTRO FERNANDES

Co-orientador:
DIOGO MENEZES FERRAZANI MATTOS

NITERÓI

2019

HELIO DO NASCIMENTO CUNHA NETO

MineCap: Detecção de Mineração de Criptomoedas em Redes Corporativas com Aprendizado de Máquina e Prevenção de Abusos com Redes

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e de Telecomunicações da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre em Engenharia Elétrica e de Telecomunicações. Área de concentração: Sistemas de Telecomunicações.

Aprovada em 08 de julho de 2019.

BANCA EXAMINADORA

Profª. NATALIA CASTRO FERNANDES, D.Sc. – Orientadora, UFF

Prof. DIOGO MENEZES FERRAZANI MATTOS, D.Sc. – Coorientador, UFF

Profª. DIANNE SCHERLY VARELA DE MEDEIROS, D.Sc. – UFF

Prof. MIGUEL ELIAS MITRE CAMPISTA, D.Sc. – UFRJ

Niterói

2019

À minha família.

Agradecimentos

Agradeço à minha família pelo apoio de sempre que, mesmo diante das dificuldades, estiveram ao meu lado para o que eu precisasse. Agradeço à minha filha pelo seu jeito alegre de ser que, mesmo nos piores dias, sempre me deu motivos para sorrir.

Aos meus orientadores, Natalia e Diogo, que sempre estiveram ao meu lado, me apoiando e me aconselhando, respondendo minhas mensagens mesmo uma da manhã. Obrigado por todo ensinamento. Hoje, eu consigo ver que evoluí muito e devo muito a vocês. Hoje, vejo que ganhei mais do que orientadores, ganhei grandes amigos.

Agradeço à Universidade Federal Fluminense que abriu suas portas e me fez sentir orgulho de fazer parte desta Universidade. Agradeço aos amigos que fiz no Laboratório MídiaCom, que sempre mantiveram um ambiente divertido e descontraído. Em especial, à professora Dianne Medeiros, que contribuiu com meu trabalho com seu vasto conhecimento e valiosos conselhos que contribuíram na minha vida acadêmica.

Agradeço a todos os meus amigos que me apoiaram, me motivaram e, mesmo quando pensei em desistir, me deram excelentes conselhos e nunca deixaram esquecer de minha motivação inicial. Dentre todos os amigos, gostaria de agradecer em especial ao Flávio Camacho que sempre me apoiou desde a época de graduação.

Agradeço ao Martin Andreoni Lopez pela sua vasta contribuição à comunidade científica, com excelentes trabalhos que serviram como inspiração. Agradeço também às orientações que o Martin me concedeu, que foram importantíssimas para a conclusão deste e de outros trabalhos acadêmicos. Gostaria de agradecer também a empresa TAESA e o programa de P&D ANEEL (PD-07130-0053/2018).

"Pouco conhecimento faz com que as pessoas se sintam orgulhosas. Muito conhecimento, que se sintam humildes."
(Leonardo da Vinci)

Resumo

A mineração não autorizada de criptomoedas implica o uso de valiosos recursos de computação, alto consumo de energia e recursos de rede. Assim, há a necessidade crescente de um sistema para proteger as informações estratégicas, recursos de rede e poder de processamento. No entanto, alguns novos fenômenos, como a mineração não autorizada de criptomoedas, não são categorizados como ataques às redes, embora prejudiquem o ambiente corporativo devido ao uso excessivo de recursos de computação, rede e energia. Existem algumas ameaças em que os invasores desenvolvem aplicativos maliciosos, como *malware* e páginas Web, para minerar criptomoedas sem o consentimento do usuário. Em ambientes corporativos, a ameaça da mineração de criptomoedas pode ser interna ou externa. A ameaça interna consiste em funcionários que usam conscientemente o poder computacional da infraestrutura da empresa para realizar mineração para fins particulares. Por outro lado, as ameaças externas consistem em entidades que exploram vulnerabilidades da infraestrutura para executar códigos de mineração de criptomoeda. Como agravante, identificar o tráfego de mineração não é uma tarefa simples, já que as características de fluxo geralmente são semelhantes às do tráfego de navegação Web, ocultando o tráfego mal-intencionado.

Nesta dissertação, é proposto um sistema em linha e dinâmico para detectar, usando técnicas de aprendizado de máquina, e bloquear fluxos de mineração não autorizada de criptomoedas em redes definidas por software. O sistema proposto baseia-se na biblioteca libpcap para capturar os pacotes de rede, Apache Kafka para transformar os pacotes recebidos em serviço de *Streaming* para ser consumido pela ferramenta de processamento em fluxo Spark *Streaming* para processamento em linha dos fluxos de rede e, ao identificar um fluxo de mineração, solicitar o bloqueio ao controlador da rede. Para classificar os fluxos de rede de uma maneira mais precisa, são testados alguns algoritmos de aprendizado de máquina para avaliar o melhor para a classificação de fluxos de mineração de criptomoedas. Posteriormente, é proposta uma nova técnica chamada de super aprendizado incremental, que é uma combinação de uma variante da técnica *super learner* e aprendizado incremental. A eficiência dessa nova técnica é testada, onde o supermodelo de aprendizado de máquina é treinado à medida que novos dados chegam, utilizando um algoritmo inteligente que seleciona as instâncias com maiores probabilidade de estarem corretas, assim fazendo o treinamento parcial do modelo, evitando o retreinamento total do modelo e o esquecimento catastrófico.

Palavras-chave: Aprendizado de Máquina, *Software Defined Network* (SDN), Criptomoeadas, *Big Data*, *Analytics*, *Super Incremental Learning*.

Abstract

Covert cryptocurrency mining implies the use of valuable computing resources, high power consumption, and network resources. Thus, there is a growing need for a mechanism to protect strategic information, network resources, and processing power. However, some new phenomena, such as the unauthorized mining of crypto-coins, are not categorized as network attacks, although they damage the corporate environment due to excessive use of computing, network, and power resources. There are some threats in which the intruders develop malicious applications, such as malware and Web pages, to mine crypto-coins without user consent. In corporate environments, the threat of cryptocurrency mining may be internal or external. The internal threat consists of employees who consciously use the computing power of the company's infrastructure to conduct mining for private purposes. On the other hand, external threats consist of entities that exploit infrastructure vulnerabilities to execute cryptocurrency mining codes. Nevertheless identifying the mining traffic is not a simple task, since the flow characteristics are generally similar to Web browsing traffic, hiding malicious traffic.

In this dissertation, an online and dynamic mechanism is proposed to detect, using machine learning techniques, and to block covert cryptocurrency mining in software-defined networks. The proposed mechanism is based on the libpcap library for capturing network packets, so Apache Kafka transforms the received packets into Streaming service to be consumed by the Spark Streaming stream processing tool for online processing of network flows and by identifying a mining flows, requests the blocking of the network controller. To classify the network flows in a more accurate way some machine learning algorithms are tested to evaluate the best for the classification of cryptocurrency mining flows. Later, the efficiency of incremental learning algorithms are tested, in incremental learning the machine learning model is trained as new data arrives, avoiding the total re-training of the model.

Palavras-chave: Aprendizado de Máquina, SDN, Criptomoeadas, Big Data, Analytics, Super Incremental Learning.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

ABS *Asynchronous Barrier Snapshotting* 45

API *Application Programming Interface* 9

ARP *Address Resolution Protocol* 19

AUC *Area Under the Curve* 54

EM *Expectation-Maximization* 14

GFS *Google File System* 38

GPS *Global Positioning System* 14

HDFS *Hadoop Distributed File System* 33

HPC *Hardware Performance Counters* 7

ICA *Independent Component Analysis* 59

ICMP *Internet Control Message Protocol* 75

IoT *Internet of Things* 16

KNN *k-Nearest Neighbors* 58

PoW *Proof of Work* 1

PSO-DS *Particle Swarm Optimization for Digital Signature* 12

PCA *Principal Component Analysis* 12

RDD *Resilient Distributed Dataset* 40

S3 *Simple Storage Service* 40

SDN *Software Defined Network* 2

SGD *Stochastic Gradient Descent* 58

SVM *Support Vector Machine* 9

SQL *Structured Query Language* 15

MLP-DS *Multilayer Perceptron for Digital Signature* 12

MLP *Multilayer Perceptron* 13

MPLS *Multiprotocol Label Switching* 28

NBI *Northbound Interfaces* 18

ONF *Open Networking Foundation* 19

ROC *Receiver Operating Characteristic* 6
REST *Representational State Transfer* 71
SBI *Southbound Interfaces* 19
SE *Seed-Expanding* 17
SSL *Secure Socket Layer* 20
IDS *Intrusion Detection System* 1
IFFD *Incremental Flexible Frequency Discretization* 65
IP *Internet Protocol* 20
IPS *Intrusion Prevention System* 1
GPU *Graphics Processing Unit* 2
CPU *Central Processing Unit* 7
GA *Genetic Algorithm* 11
DDoS *Distributed Denial of Service* 12
DoS *Denial of Service* 10
MAC *Media Access Control* 20
NETCONF *Network Configuration* 20
NIDS *Network Intrusion Detection Systems* 11
OCSVM *One-Class Support Vector Machine* 12
OF-CONFIG *OpenFlow Configuration Protocol* 20
OPFC *Optimum-Path Forest Clustering* 16
VFDT *Very Fast Decision Tree* 63
VLAN *Virtual Local Area Network* 19

Sumário

Capítulo 1

Introdução

As redes corporativas são alvos constantes de uma variedade de ameaças [44, 77]. Assim, há a necessidade crescente de ferramentas para proteger as informações estratégicas, recursos de rede e poder de processamento. Existem diversos tipos de sistemas que protegem as redes de computadores de ameaças como *Firewalls*, *Intrusion Detection System* (IDS), *Intrusion Prevention System* (IPS), Antivírus, etc. Existem algumas propostas na literatura que utilizam técnicas de aprendizado de máquina e de *Big Data Analytics* para realizar classificação de fluxos de rede em tráfego normal e tráfego malicioso, com a intenção de identificar ataques desconhecidos [7]. Essas técnicas visam identificar anomalias de rede e novos ataques intitulados de vulnerabilidades *zero-day*. Existem duas categorias mais conhecidas para detecção de *Malwares*, técnicas de detecção baseadas em assinatura e baseadas em anomalia [43]. A detecção baseada em assinatura emprega um conjunto de assinaturas predefinidas para *Malwares* já conhecidos para determinar se o software possui algum código malicioso. A principal limitação da abordagem baseada em assinatura é sua falha em detectar ataques *zero day*, que são ameaças emergentes e até então desconhecidas do sistema de detecção de *Malware*. Já os sistemas baseados em detecção de anomalias visam identificar comportamentos anormais na rede que podem ou não ser uma ameaça. Contudo, existem diversos trabalhos na literatura que utilizam algoritmos de aprendizagem supervisionadas para treinar um modelo a identificar *malware* e tráfegos anômalos.

Com a popularização das criptomoedas, como por exemplo bitcoin e ethereum, pessoas mal-intencionadas exploram vulnerabilidades da rede e de outros dispositivos dentro de redes corporativas, onde esse processo geralmente é proibido, para realizarem o processo de mineração sem o consentimento do usuário. A maioria dos processos de mineração utilizam o protocolo de prova de trabalho, do inglês *Proof of Work* (PoW). No protocolo PoW, o minerador encontra um novo bloco, que pode consistir de uma nova transição utilizando uma criptomoeda que deve ser adicionada a cadeia de blocos. Para isso, o minerados deve resolver uma nova função criptografada que combine com a do bloco anterior. A prova de trabalho consiste em resolver essa função. Verificar se o resultado da função está correto é um processo fácil, porém o processo inverso é muito difícil. Por isso, muitos mineradores

utilizam hardwares específicos ou até mesmo *Graphics Processing Unit* (GPU) para essa atividade, pois a mineração requer um grande esforço computacional e os mineradores são recompensados com criptomoedas [54].

Como, de fato, muitas pessoas e empresas estão ganhando dinheiro com mineração, esse processo não é categorizado como ameaça às redes, mas prejudicam o ambiente corporativo devido ao uso excessivo de recursos de computação, rede e energia [23, 88]. Existem algumas ameaças, categorizadas como *cryptojacking*, onde os invasores desenvolvem aplicativos maliciosos, como *malware* ou são embutidas em páginas Web, para minerar criptomoedas sem o consentimento do usuário. Em redes corporativas, a ameaça da mineração de criptomoedas sem consentimento pode ser categorizada como interna ou externa:

- As ameaças internas consistem em funcionários utilizarem conscientemente o poder computacional da infraestrutura da empresa para realizar mineração para fins particulares; na maior parte das vezes, são funcionários que são responsáveis pela infraestrutura da empresa.
- As ameaças externas consistem em entidades que exploram vulnerabilidades da infraestrutura, como, por exemplo, invadir roteadores para instalar códigos maliciosos para executar códigos de mineração de criptomoeda ou até mesmo embutir códigos de mineração em páginas WEB.

Portanto, identificar o tráfego de mineração não é uma tarefa simples, já que as características de fluxo geralmente são semelhantes às do tráfego de navegação Web, ocultando o tráfego mal-intencionado.

Há uma clara necessidade de um sistema para detectar e impedir o tráfego não autorizado de mineração em ambientes em que as políticas de rede proíbem a atividade de mineração de criptomoeda. Nesta dissertação, é proposto o MineCap*, um sistema que utiliza algoritmos de aprendizado de máquina para realizar a classificação em linha (*online*) do tráfego de mineração de criptomoeda em redes definida por *software*, *Software Defined Network* (SDN). Para desenvolver a ferramenta, avalia-se o desempenho de quatro algoritmos de classificação: Floresta Aleatória, *Naive Bayes*, Regressão Logística e *Gradient Booster Tree*. Posteriormente, os melhores classificadores são escolhidos para comporem uma nova técnica chamada *super incremental learning*. Nesta nova técnica, utiliza-se mais de um classificador para realizar o treinamento e a previsão, sendo que a saída de um classificador server como entrada para um outro classificador. O classificador que recebe como entrada as probabilidades dos outros classificadores é chamado de *super learner*. No cenário desta dissertação, o *super learner* é uma rede neural incremental, com a qual é criado um algoritmo para selecionar as melhores instâncias para o aprendizado incremental. O MineCap [20] é implementado utilizando a ferramenta *Apache Spark*, usando as bibliotecas *Spark Streaming* para manipular o fluxo de dados e a MLlib para

*Disponível em https://www.midiacom.uff.br/minecap/.

fornecer os algoritmos de aprendizado de máquina para realizar a classificação em linha. A avaliação do protótipo do MineCap foi realizada em uma rede emulada pela plataforma Mininet[†], utilizando o controlador SDN Ryu[‡] com a API REST habilitada para instalar regras de bloqueio de fluxos.

Embora os trabalhos anteriores se concentrem em identificar o tráfego de mineração usando o aprendizado de máquina [88, 50], esses trabalhos carecem de integração entre a classificação e a rede e, também, faltam contramedidas ativas para descartar os fluxos de mineração. Na rede definida por software, o plano de controle e encaminhamento é desacoplado e introduz uma nova entidade, o controlador SDN [12], responsável por executar todas as ações de controle na rede. O controlador SDN permite que a rede se torne programável e que os aplicativos de controle interajam diretamente com os elementos de rede [67]. As redes definidas por *software* necessitam de um controlador logicamente centralizado, mas a implementação do controlador pode ser fisicamente distribuída [66]. O MineCap é baseado neste paradigma de rede para emitir novas políticas para eliminar o tráfego de mineração de criptomoeda diretamente nos elementos de rede.

1.1 Motivação

A mineração não autorizada de criptomoedas torna-se um inconveniente, pois o processo de mineração acaba onerando a rede, recursos computacionais e elétricos e os principais sistemas de detecção de intrusão e *firewalls* não possuem, até o momento, uma prevenção para esse problema. Na literatura, os principais trabalhos que visam identificar a mineração não autorizada de criptomoedas não apresentam um sistema eficiente que proteja toda a rede de computadores da mineração de criptomoedas. Como consequência, observa-se atualmente inúmeros exemplos de ataques de mineração não autorizada. Por exemplo, em 2018, um hacker invadiu milhares de roteadores para realizar mineração não autorizada da criptomoeda Monero, utilizando o código CoinHive para realizar o processo de mineração[§]. O *site* PirateBay também foi identificado utilizando o código CoinHive para realizar mineração sem o consentimento de seus usuários. Nesse caso, ao acessar o *site* do PirateBay, o usuário acaba executando um código de mineração escrito em JavaScript[¶]. Existe também a possibilidade de usuários mal-intencionados se aproveitarem da vulnerabilidade de um *site* com bastante acesso e implantar um código de mineração. Esse caso aconteceu em um dos *sites* do governo de São Paulo, o Portal do Cidadão. Nesse caso, o código malicioso foi

[†]Disponível em https://github.com/mininet/mininet.

[‡]Disponível em https://osrg.github.io/ryu/.

[§]Notícia retirada do site: https://g1.globo.com/economia/tecnologia/blog/altieres-rohr/post/2018/08/02/hackers-atacam-roteadores-mikrotik-no-brasil-para-minerar-criptomoedas-na-web.ghtml.

[¶]Notícia retirada do site: https://www.bbc.com/news/technology-41306384.

retirado da página assim que os administradores foram notificados[‖]. O código inserido no Portal do Cidadão pode ser visto na Figura 1.1.

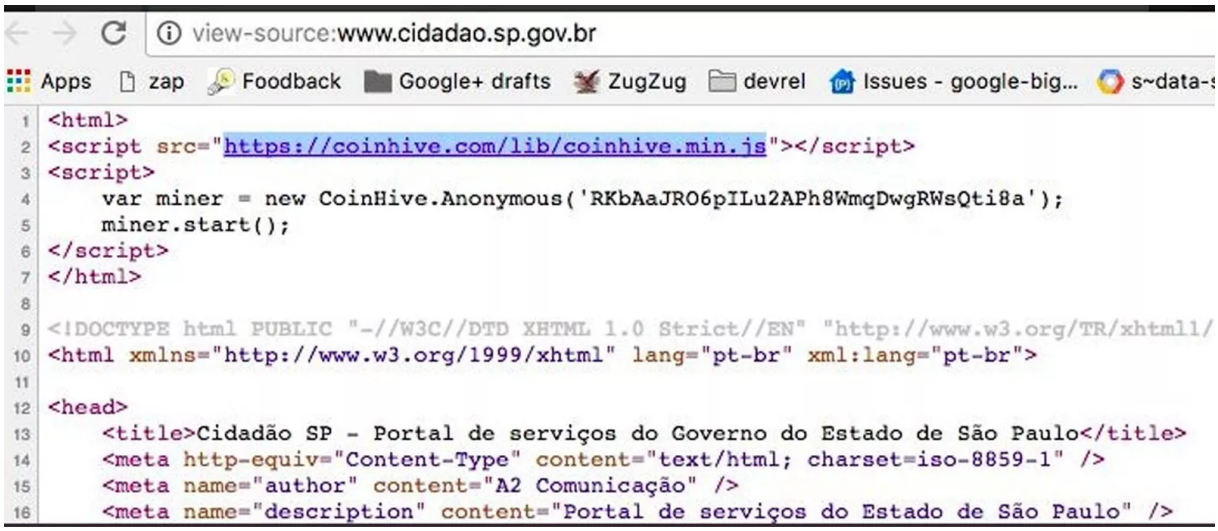


Figura 1.1: Código fonte do *site* Portal do cidadão, destacando o *script* de mineração.

Os exemplos citados anteriormente são categorizados como ameaças externas, onde uma entidade fora da rede de computadores de aproveita de vulnerabilidades da rede para realizar mineração. A outra categoria de mineração não autorizada é a interna, onde pessoas dentro da rede executam códigos de mineração, intencionalmente, mas sem permissão. Com isso, é necessário um sistema que preencha os seguintes requisitos:

- Realize a proteção de toda rede interna - Como dito anteriormente, os principais trabalhos da literatura focam em identificar a mineração não autorizada, porém deixam de lados a infraestrutura de rede. É necessário um sistema que proteja toda a rede da mineração não autorizada.

- Proteção em linha de tolerante a falha - O MineCap utiliza o ecossistema Apache Spark para processamento de dados em linha, que é um processamento próximo do real. O Apache Spark fornece ao sistema a possibilidade de computação distribuída com sistema de tolerância a falha melhor do que outras ferramentas como Apache Flink e Apache Storm [58].

- Bloqueio de fluxo de mineração: Os fluxos de mineração devem ser identificados e bloqueados com a menor taxa de falsos positivos possível, assim evitando bloqueio de tráfego normal, o que comprometeria o bom funcionamento da rede.

[‖]Notícia retirada do site: https://g1.globo.com/tecnologia/noticia/site-do-governo-de-sp-usou-computador-de-visitante-para-minerar-moeda-virtual.ghtml.

1.2 Objetivos

O objetivo principal dessa dissertação é desenvolver um sistema de detecção e prevenção de mineração de criptomoedas capaz de identificar esses fluxos e solicitar o bloqueio para o controlador SDN, que é o responsável por instalar fluxos de rede nos *switches* OpenFlow de maneira proativa e/ou reativa. Durante o desenvolvimento do sistema MineCap, foi proposta uma nova técnica chamada de super aprendizado incremental, que mescla uma variante da técnica super *learner* [94] com aprendizado incremental. O super aprendizado incremental foi desenvolvido para evitar o desvio de conceito, problema comum em sistemas de processamento de grandes massas de dados em linha. Foram testados quatro algoritmos de aprendizado de máquina e os melhores são escolhidos para fazer parte do super aprendizado incremental. Os principais tópicos abordados nesta dissertação são as técnicas de processamento de grandes massas de dados, pré-processamento de dados, aprendizado de máquina, redes definidas por software e, finalmente, a proposta de um sistema capaz de identificar e bloquear fluxos de mineração.

1.3 Contribuições

O principal foco dessa dissertação de mestrado é o sistema de detecção de mineração de criptomoedas MineCap. Para desenvolver o MineCap, houve um trabalho de pesquisa para escolher quais as melhores ferramentas utilizadas atualmente para detecção de anomalias de rede. Foram feitos, também, testes em algoritmos de aprendizado de máquina para escolher qual o melhor classificador para a detecção de mineração de criptomoedas.

Assim, as principais contribuições dessa dissertação são:

- Um sistema de detecção de mineração de criptomoedas, que pode ser instalado em qualquer *appliance* que utilize o sistema operacional Linux. O sistema MineCap pode solicitar bloqueio de fluxo para qualquer controlador SDN que possua uma interface REST.
- Estado da arte dos algoritmos de aprendizado de máquina e processamento de grandes massas de dados para detecção de anomalias de rede.
- Análise de algoritmos de aprendizado incremental e avaliação de seus desempenhos utilizando uma ferramenta de processamento em fluxo.
- Mecanismo de bloqueio de fluxos de mineração de criptomoedas utilizando o protocolo OpenFlow para instalar fluxos de bloqueio.
- Avaliação dos algoritmos mais utilizados na detecção de anomalias de rede utilizando os métodos para avaliação de classificadores de aprendizado supervisionado

como precisão, sensibilidade, especificidade, acurácia e curva *Receiver Operating Characteristic* (ROC).

- A proposta e a avaliação de uma nova técnica de aprendizado de máquina chamada de super aprendizado incremental, que visa, além de ter um super modelo alimentado pelos melhores classificadores de um determinado problema, conseguir aprender com novos dados recebidos em linha.

1.4 Estrutura do Texto

O restante do texto está organizado em nove capítulos. No Capítulo 2, são abordados os trabalhos relacionados. No Capítulo 4, são apresentados os conceitos de processamento de grandes massas de dados, comparando processamento em lote com processamento em fluxo e abordando as principais ferramentas. No Capítulo 5, é feita a análise das principais técnicas de pré-processamento de dados. No Capítulo 6, são introduzidos os conceitos sobre aprendizado de máquina. No Capítulo 3, são abordadas as redes definidas por *software* e seu principal protocolo, o OpenFlow. No Capítulos 7, é descrita a nova técnica proposta chamada de super aprendizado incremental, onde é proposta uma variante da técnica super *learner* utilizando um modelo de aprendizado incremental. No Capítulo 8, é descrito o funcionamento do sistema MineCap e sua avaliação. O Capítulo 9 conclui essa dissertação e apresenta alguns trabalhos futuros.

Capítulo 2

Trabalhos Relacionados

Neste capitulo, são abordados os trabalhos relacionados, que são divididos em duas seções. A primeira é sobre ferramentas que identificam mineração de criptomoedas e a segunda é sobre ferramentas que utilizam aprendizado de máquina e *Big Data Analytics* para detecção de anomalias de rede. Durante a pesquisa, todos os trabalhos encontrados protegiam apenas um único *host* da mineração não autorizada de criptomoedas. Um dos objetivos dessa dissertação é levar essa proteção para toda a rede, utilizando a abordagem de detecção de anomalias de redes com *Big Data Analytics* e aprendizado de máquina.

2.1 Detecção de Mineração de Criptomoedas

Tahir *et al.* propõem a ferramenta MineGuard para detectar em tempo real o comportamento de processos de mineração em máquinas virtuais [88]. MineGuard utiliza contadores de desempenho *Hardware Performance Counters* (HPC), um conjunto de registradores integrados em processadores modernos, para armazenar as contagens de atividades relacionadas ao *hardware*. Os contadores permitem rastrear com precisão as operações de mineração de baixo nível ou eventos dentro da *Central Processing Unit* (CPU) e GPU com sobrecarga mínima, dando à ferramenta a capacidade de detectar com precisão, em tempo real, se uma máquina virtual está tentando minerar criptomoedas. A MineGuard baseia-se na observação de que, para minerar criptomoedas, é necessário executar repetidamente o algoritmo de PoW.

O MineSweeper [50] usa a URL do sítio *Web* como entrada para verificar se há algum *software* de mineração de criptomoedas oculto. O MineSweeper usa uma métrica para identificar funções criptográficas, medindo o número de operações executadas por um aplicativo. Os autores analisam o algoritmo CryptoNight e suas variantes, mas argumentam que adicionar outros algoritmos é uma tarefa trivial. O MineSweeper concentra-se na mineração embarcada em páginas Web e seu principal objetivo é identificar um novo ataque, o *drive-by mining*, no qual um sítio infectado executa secretamente código JavaScript ou

um módulo WebAssembly no navegador do usuário para minerar criptomoedas sem o seu consentimento. Diferentemente do MineSweeper, o MineCap concentra-se em detectar e bloquear tráfego de mineração de criptomoeda na rede.

Rüth *et al.* [81] propõem utilizar a mineração de criptomoedas em navegadores, com o consentimento do usuário, como uma alternativa ao sistema de anúncio em websites. Nem todas as criptomoedas são igualmente adequadas para a mineração em navegadores. O desequilíbrio de hardware e a consequente alta dificuldade de mineração do Bitcoin torna ineficiente sua mineração no navegador e motiva o uso, por exemplo, do Monero como uma moeda alternativa que pode ser eficientemente extraída em CPU e, portanto, em navegadores. O Monero foi adotado por sites e até mesmo entre *botnets* para minerar em milhões de hosts comprometidos. A principal proposta do artigo é propor *Coinhives*, *script* que os administradores de websites podem usar em seus sites para minerar criptomoeda Monero para o administrador do site utilizando os recursos do visitante, como um serviço a ser utilizado que fornece uma estrutura para incorporar um minerador da Monero em um site. Embora essas estruturas permitam a mineração sem o conhecimento dos usuários, outros serviços podem solicitar aos usuários o consentimento deles para fazê-lo como alternativa à exibição de anúncios. Os autores também avaliaram que a lista pública NoCoin, que contêm uma lista de sites que possuem códigos de mineração, como insuficiente para detectar a ampla gama de sites que possuem códigos de mineração. Apesar do artigo não propor um mecanismo de bloqueio de mineração não autorizada de criptomoedas, eles avaliam o mecanismo de uma lista pública que listam sites com código de mineração.

Em Eskandari *et al.* [26], é analisada a tendência de mineração de criptomoedas em navegadores, em particular, a mineração do Monero através *Coinhive* e semelhantes. Neste artigo, é feito um panorama realizando algumas medições para estabelecer sua prevalência e lucratividade. Foi feito um arcabouço ético para considerar se essa prática deve ser classificada como um ataque ou oportunidade de negócio. São feitas também sugestões para a detecção, mitigação e/ou prevenção de mineração baseada em navegador sem consentimento do usuário. Os autores classificam *cryptojacking* em sítios da web em três classes: i) O uso de *cryptojacking* em um site violado, ou seja, uma terceira entidade vê uma falha no site para implantar códigos de mineração, como no espaço de anúncio por exemplo. ii) o uso de *cryptojacking* pelo dono do site obtendo o consentimento do usuário, porém nem sempre o usuário tem a ciência do que ele está concordando e o que ele estará recebendo em troca e iii) o uso de *cryptojacking* pelo dono do site sem obter o consentimento do usuário, que é provavelmente o mais antiético. Para solucionar o problema de mineração via navegador sem o consentimento do usuário, os autores propõem o uso de aplicativos para navegadores que detectam o uso de códigos de mineração. Alguns navegadores, como por exemplo o Opera, utilizam o filtro público *NoCoin* para realizar o bloqueio de sites que fazem uso da prática de *cryptojacking*. Outros medem o uso deCPU e alertam os usuário sobre o alto uso desse recurso por um determinado sítio da web.

Wang *et al.* [97] propõem o SEISMIC (*SEcure In-lined Script Monitors for Interrupting*

Cryptojacks), que é um programa que modifica automaticamente os programas binários Wasm para que eles se auto-projetem enquanto são executados, detectando atividades de mineração. Segundo os autores, os novos códigos de mineração estão utilizando WebAssembly (Wasm), que é uma nova linguagem de *bytecode* para navegadores Web que oferece computação mais rápida e eficiente do que as linguagens de *script* anteriores, como o JavaScript. Com a implementação de algoritmos mineração em Wasm, os mineradores podem fazer uso mais eficiente dos recursos de computação do cliente para gerar uma receita maior. Então, o SEISMIC oferece um mecanismo de detecção de *crypjacking* baseado em *scripts* Wasm que é mais robusto e acurado do que as atuais defesas de detecção empregadas por produtos antivírus e *plugins* de navegadores. O SEISMIC é um programa que deve ser instalado em uma máquina e só pode defender o *host* onde está instalado, diferente do MineCap, que pode proteger uma rede inteira de diferentes tipos de *cryptojacking*, não somente os baseados na Web.

Dao *et al.* [21] fazem uma análise de uso abusivo de recurso por páginas web do TOP 150.000 da lista Alexa, uma lista contendo os sites mais acessados do mundo. Foi feita uma análise dos sites que consomem recursos utilizando códigos maliciosos e códigos de mineração. Os autores avaliam possíveis contramedidas para esses tipos de ataque. Os autores categorizam por país e categorias, e identificaram que a China é o país com a maior quantidade de sites com códigos maliciosos e os Estados Unidos é o país com a maior quantidade de códigos de mineração em navegadores. Nos Estados Unidos, sites adultos contém maior índice de códigos maliciosos, enquanto sites de tecnologia da informação possuem maior índice de códigos de mineração de criptomoedas. Os autores defendem que as listas de filtro de sites com códigos de mineração pode mitigar a maioria dos sites que realizam *cryptojacking* baseados em páginas web, o que é divergente com a análise de Rüth *et al.*

No trabalho de Liu *et al.* [56], são utilizadas redes neurais recorrentes para identificar mineração de criptomoedas em navegadores web. Os autores modificaram o código do kernel do Chrome para implantar o detector de *cryptojacking*. Denominado BMDetector, um protótipo de detecção de mineração silencioso baseado em navegador foi projetado e implementado. Com 1159 amostras detectadas e analisadas, o protótipo criado pelos autores possui excelentes resultados com uma acurácia média de 98%.

Em Rodriguez *et al.* [79], é criado um mecanismo de detecção de código de mineração em navegadores. Para isso, os autores utilizam o algoritmo de aprendizado de máquina *Support Vector Machine* (SVM), que, segundo os autores, é prevalente para lidar com problemas de segurança, pois, ao contrário de outros métodos de classificação que minimizam a perda empírica, o SVM tenta minimizar a perda de generalização. Os autores também utilizaram a lista Alexa com os TOP 330.500 sites mais visitados. Foram utilizados navegadores reais para visitar sites enquanto eram monitoradas chamadas da *Application Programming Interface* (API) relacionadas ao consumo de recursos de CPU do navegador. Além disso, as técnicas de detecção utilizadas pelos autores generalizam bem e classificam amostras

inéditas. O mecanismo funciona como uma API do navegador que monitora os recursos e as aplicações que são chamadas pelo *site*. O mecanismo possui resultados superiores aos das listas negras de sites que utilizam códigos de mineração, mas ainda assim como todos os outros trabalhos, funciona apenas no *host* em que a API é instalada.

Hong *et al.* [40] propõem o CMTracker, um detector de mineração de criptomoedas baseado em comportamento com dois perfis de tempo de execução para rastrear automaticamente *scripts* de mineração e seus domínios relacionados. O mecanismo proposto identificou com sucesso 2.770 amostras exclusivas de *cryptojacking* a partir de 853.936 páginas populares da web, incluindo 868 entre as 100 mil principais da lista Alexa. Primeiro, foram coletadas 853.936 amostras de páginas da web contendo *cryptojacking* como o conjunto de dados para o estudo de medição. Depois, foram localizadas as páginas de mineração de criptomoedas aproveitando dois perfis baseados em comportamento. O *design* do CMTracker é para detectar páginas Web que contenham *cryptojacking* aprendidas pelo conjunto de dados. Os autores não garantem o bloqueio de todas as técnicas de evasão existentes ou futuras.

2.2 Detecção de Anomalias de Rede

Andreoni *et al.* [7] propõem uma ferramenta em linha para a detecção e prevenção de ataques de rede implementada como uma função virtual de rede, chamada CATRACA. Assim como o MineCap, a ferramenta proposta utiliza um sistema de processamento de fluxo de grande massa de dados Apache Spark, fornecendo um serviço de detecção de ameaças em tempo real. A ferramenta opera em dois modos: *online* ou *offline*. O processo de detecção inclui algoritmos de seleção de características e aprendizado de máquina para discriminar tráfego normal, *Denial of Service* (DoS) e varredura de portas. Em uma detecção de ameaça, o sistema pode reagir prontamente e criar regras de bloqueio em um firewall. O MineCap usa uma abordagem semelhante, mas integra a capacidade de bloqueio de fluxos utilizando o protocolo OpenFlow, sem a necessidade de usar outras ferramentas. A CATRACA fornece uma interface gráfica que permite a visualização em tempo real da atividade da rede e dos ataques que ocorrem na rede. A camada de captura do MineCap foi baseada em funções similares às da ferramenta CATRACA. O MineCap, no entanto, diferencia-se por focar na detecção de mineração de criptomoeda, enquanto a ferramenta CATRACA foca na detecção de ataques de negação de serviço e detecção de anomalias de rede.

Tang *et al.* [89] propõem uma nova abordagem utilizando aprendizagem profunda para detecção de intrusos em uma rede definida por software. A aprendizagem profunda é capaz de encontrar correlações nos dados, diferente dos outros métodos de aprendizado de máquina, por isso é um método promissor para a próxima geração de detecção de intrusão. Para realizar o treinamento, os autores utilizaram o conjunto de dados NSL-KDD, que

é bastante antigo e não representa perfeitamente as redes reais existentes. Para realizar o treinamento, os autores escolheram seis dos quarenta e uma características existentes no conjunto de dados e não citam o porquê da escolha dessas características. O módulo *Network Intrusion Detection Systems* (NIDS) proposto foi instalado no controlador SDN em tempo real, mas o artigo carece de explicações de como esses fluxos foram obtidos e tratados, se foi utilizado uma ferramenta de processamento de fluxo ou se os fluxos foram gravados em um arquivo para posterior classificação.

Shon *et al.* [83] propõem um arcabouço para detecção de anomalias de rede utilizando SVM e *Genetic Algorithm* (GA). Para realizar a classificação do tráfego, o arcabouço realiza a seleção de características utilizando algoritmos genéticos, a fim de remover as características menos importantes, assim diminuindo o custo computacional e melhorando a classificação. Os autores utilizaram tráfego real para efetuar os testes e obtiveram bons resultados comparados a outras soluções de NIDS, como Snort e Bro. O artigo carece de resultados comuns entre artigos de aprendizado de máquina como precisão, sensibilidade, curva ROC, e outros.

No trabalho de Tsai *et al.* [92], também procura-se fazer uma revisão do uso de aprendizado de máquina para detecção de intrusos. O objetivo dos autores foi revisar cinquenta e cinco artigos e examinar quais técnicas obtiveram melhores resultados. Foi considerado um grande número de técnicas de aprendizado de máquinas utilizadas no domínio de detecção de intrusão para a revisão, incluindo classificadores simples e híbridos. O artigo apresenta o grande uso dos algoritmos SVM, redes neurais artificiais, árvore de decisão e k-vizinhos mais próximos para classificação de anomalias de rede e propõe que a criação de classificadores mais sofisticados através da combinação de classificadores é bastante valiosa para se obter melhores resultados.

Em Zhao *et al.* [104], é proposto um sistema que analisa e classifica em tempo real todo o tráfego de rede do campus da universidade de Missouri na cidade de Kansas. O sistema conta com as ferramentas Apache Hadoop, Apache Kafka e Apache Storm para lidar com a grande massa de dados em tempo real. O sistema utiliza a ferramenta WEKA, que implementa técnicas de aprendizado de máquina para a classificação do tráfego. O trabalho carece de mais resultados, como precisão, sensibilidade e curva ROC, que são resultados muito utilizados em artigos de aprendizado de máquina para avaliar o desempenho do classificador utilizado nos experimentos.

Lin *et al.* [55] propõem um novo tipo de representação de características a fim de reduzir o esforço computacional e obter uma melhor classificação. No artigo, os autores combinam as técnicas K-means e KNN criando uma nova técnica chamada de CANN que, dado um conjunto de dados, utiliza o algoritmo de aglomeração k-means para extrair os centros do cluster de cada categoria predefinida. Em seguida, o vizinho mais próximo de cada amostra de dados no mesmo cluster é identificado. Após isso, calcula-se a soma da distância entre uma amostra de dados específica e os centros do cluster e a distância entre essa amostra de

dados e seu vizinho mais próximo. Isso resulta em um novo recurso baseado em distância que representa os dados no conjunto de dados fornecido. Consequentemente, um novo conjunto de dados contendo apenas uma dimensão é usado para o método k-vizinhos mais próximos. O CANN torna-se uma alternativa ao *Principal Component Analysis* (PCA), que é frequentemente utilizado para redução de dimensão. O CANN foi testado utilizando o conjunto de dados KDD.

Maglaras e Jiang [64] propõem um método de detecção de intrusos em um sistema SCALA. Para minimizar as desvantagens conhecidas em utilizar ferramentas de aprendizado de máquina em sistema de detecção de intrusão, os autores utilizam uma abordagem inteligente baseada nos princípios do *One-Class Support Vector Machine* (OCSVM). O OCSVM é uma extensão natural do algoritmo de vetor de suporte para o caso de dados não rotulados, especialmente para detecção de *outliers*. O algoritmo OCSVM mapeia os dados de entrada em um espaço de características de alta dimensão (via *kernel*) e, de forma iterativa, localiza o hiperplano de margem máxima que melhor separa os dados de treinamento da origem. O método proposto trabalha em uma forma *offline*, ou seja, há um atraso na coleta de informações e classificação, e não foram apresentados resultados como precisão, sensibilidade, curva ROC, e outros.

Assis *et al.* [22] apresentam um sistema de defesa contra ataques de *Distributed Denial of Service* (DDoS) e varredura de portas em redes definidas por software. O sistema apresentado pelos autores é executado no próprio controlador, o que pode ocasionar a sobrecarga. Para a detecção, os autores comparam três abordagens diferentes, otimização por enxame de partículas, redes neurais e transformada discreta de *wavelet*. O sistema possui três módulos: um para a Detecção do ataque, outro para a identificação e coleta de informações e outro para mitigação do ataque. O sistema proposto pelos autores captura fluxos, que são armazenados em um arquivo para posterior classificação. O atraso desse processo torna o sistema proposto inferior aos outro que utilizam ferramenta de processamento de fluxo que faz a classificação em tempo real. Os autores utilizam perceptron de múltiplas camadas para assinaturas digitais *Multilayer Perceptron for Digital Signature* (MLP-DS) para aprendizado supervisionado e otimização por enxame de partículas para assinaturas digitais *Particle Swarm Optimization for Digital Signature* (PSO-DS) e *WaveDetect* para aprendizado não supervisionado, ou seja, quando não há um conjunto de dados para treinamento.

Lobato *et al.* [57] propõem uma arquitetura adaptativa de detecção de ameaças em tempo real composta pelo uso de sistemas processamento de fluxo distribuído com dados coletados de *honeypots* para treinamento de algoritmos de aprendizado de máquina em tempo real. É considerado todo o tráfego do *honeypot* como malicioso, pois não há recursos úteis nessa rede, então os dados coletados são utilizados para atualizar os algoritmos de detecção de ameaças. A arquitetura se adapta às mudanças de comportamento do invasor e aprende ataques de dia zero(*zero-day*). A arquitetura proposta apresenta tanto a classificação de ameaças supervisionada, ataques conhecidos, em linha(*on-line*) quanto

a detecção de anomalias não supervisionadas, novos ataques. Os autores utilizaram o sistema de processamento de fluxo Storm. Os autores avaliaram o desempenho de quatro esquemas de detecção usando conjuntos de dados reais, um criado em laboratório e o outro contendo dados de uma das principais operadoras de rede. Os resultados apresentados pelos autores mostram que os esquemas de classificação em linha são capazes de aprender ameaças desconhecidas utilizando dados do *honeypot* e manter altas taxas de precisão, superiores a 90%, conforme os fluxos de tráfego chegam.

Em Wai *et al.* [95] é proposto o uso de técnicas de aprendizado de máquina para análise de tráfego de rede para a detecção de tráfego de *botnet*. Os autores melhoraram os componentes de um arcabouço de detecção existente com as técnicas de aprendizado de máquina para automatizar seus processos e melhorar o desempenho ao mesmo tempo. Portanto, o objetivo do trabalho é explorar a eficácia da automação de técnicas de detecção de anomalias de rede que dependem do uso de processos manuais e limiares humanos com aprendizado de máquina. Neste trabalho, são propostos dois métodos para automatizar uma estrutura manual existente que detecta o tráfego de *botnets* com técnicas de aprendizado de máquina, incluindo árvores de decisão e *Multilayer Perceptron* (MLP). No método um, foi criado um conjunto de características composto de parâmetros subjacentes obtidos a partir da análise de regressão na segunda fase do arcabouço original. Portanto, nesta modificação, a aplicação de técnicas de regressão nas características básicas extraídas no primeiro estágio pode ser tratada como uma forma de engenharia de características. Com esse novo conjunto de características, foram avaliados Máquina de Vetor Suporte, Árvore de Decisão e Floresta Aleatória. No método dois, é utilizado MLP para realizar a engenharia de características e, por fim, o conjunto de dados modificados é entregue para classificação, avaliando os algoritmos MLP e Árvore de Decisão.

Habeeb *et al.* [35] realizam um levantamento apontando as principais questões e desafios da detecção de anomalias em tempo real. Os autores pesquisaram o estado da arte das tecnologias de processamento de grandes massas de dados em tempo real de última geração relacionadas à detecção de anomalias e as características vitais dos algoritmos de aprendizado de máquina associados. Os autores propõem uma taxonomia dos algoritmos de processamento de dados em grande escala, detecção anomalia e algoritmos de aprendizado de máquina em tempo real, seguidos pela revisão de tecnologias de processamento de grande massa de dados. Este trabalho destaca os desafios de pesquisa e propõe um guia para pesquisas futuras. Os autores, além de analisar as principais ferramentas de processamento de dados para detecção de anomalias e analisar os principais algoritmos de aprendizado de máquina, também propõem métodos para avaliar o possível arcabouço para detecção de anomalias na rede. O artigo serve como um guia para construir um arcabouço de detecção de anomalias de rede utilizando *big data analytics*.

Gonçalves *et al.* [34] apresentam uma abordagem para avaliar a segurança da infraestrutura usando seus *logs*. Para tanto, os autores utilizaram dados de uma rede de telecomunicações real. Foram utilizadas técnicas de aprendizado de máquina e mineração

de dados para analisar os dados e descobrir os *hosts* com comportamento inadequado. A abordagem utilizada combina mineração de dados e aprendizado de máquina supervisionado e não supervisionado para tornar o processo de detecção o mais automático possível, embora, utilizando essa abordagem, as pessoas não são totalmente removidas do processo. O sistema proposto analisa os *logs* de *hosts*, *firewalls*, roteadores, etc. O protótipo possui dois componentes principais. Primeiro, a normalização e a extração de características são executadas por um conjunto de tarefas do Hadoop MapReduce. Em seguida, o agrupamento e a classificação são feitos usando o WEKA, um software de mineração de dados bem conhecido escrito em Java. Para realizar a aprendizagem não supervisionada e separar as saídas em dados agrupados, foi utilizado o *Expectation-Maximization* (EM) e para a classificação, os autores utilizaram *Naive Bayes*. Os resultados obtidos não são precisos o suficiente para executar ações automáticas, como, por exemplo, colocar em quarentena os *hosts* em um *cluster* classificado como suspeito. No entanto, ele extrai informações de segurança relevantes dos *logs* que, de outra forma, não são diretamente observáveis.

McNeil *et al.* [70] propõem o SCREDENT, acrônimo para *Scalable Real-time Anomalies Detection and Notification of Targeted Malware in Mobile Devices* (Detecção e Notificação Escalonada de Anomalias em Tempo Real de *Malware* Direcionado em Dispositivos Móveis), para fornecer um sistema escalável para classificar e detectar *malware* em tempo real. O SCREDENT aproveita a tecnologia de contêiner para realizar análises dinâmicas e permitir modularidade à medida que a tecnologia de emulação melhora. O SCREDENT usa princípios de notificação adaptáveis baseados em localização para criar uma cerca geográfica que avise os usuários sobre ataques maliciosos. O SCREDENT integra a criação de perfis de grupos de usuários, o que ajuda a automatizar a análise dinâmica orientada por comportamento na detecção de *malware*. A ferramenta utiliza SVM para a classificação e k-means para o agrupamento das diferente ameaças. O modelo SVM é treinado na nuvem e enviado ao *smartphone* para classificação. Para lidar com dados em tempo real, a ferramenta utiliza Apache Spark e MongoDB. Para a ferramenta funcionar corretamente, é necessária conexão WiFi, para análises profundas e atualização do modelo, e *Global Positioning System* (GPS), para a notificação baseada na localização.

Singh *et al.* [84] fazem o uso de ferramentas de código aberto, como Hadoop, Hive e Mahout, para fornecer uma implementação escalável do sistema de detecção de intrusão em tempo quase real. A implementação é usada para detectar ataques de *botnets* ponto a ponto usando a abordagem de aprendizado de máquina. As tecnologias utilizadas neste arcabouço são o libpcap, o Hadoop, MapReduce e o Mahout. Tshark é usado para extrair os campos requeridos dos pacotes. Estes são necessários para a geração do conjunto de recursos para o Módulo de Aprendizado de Máquina. O TShark é um analisador de protocolo de rede que usa a biblioteca libpcap e captura dados de pacote de uma rede ativa. Ele também permite imprimir uma forma decodificada e personalizável dos pacotes capturados na saída padrão ou em um arquivo. Após a extração da informação desejada é utilizado o MapReduce para a extração de características. Isso é feito usando o Apache Hive, que fornece um

mecanismo para consultar os dados usando uma linguagem semelhante a *Structured Query Language* (SQL) chamada HiveQL. Uma grande desvantagem do arcabouço é que a análise é feita de modo *offline*. O arcabouço utiliza floresta aleatória, porém foram realizados testes com outros classificadores que foram inferiores ao algoritmo escolhido.

Em Rathore *et al.* [78] é apresentado um IDS em tempo real, capaz de funcionar em ambiente de processamento de grande massa de dados. O sistema possui quatro camadas, consistindo nas camadas de captura, filtragem e balanceamento de carga, processamento e tomada de decisão. É utilizada engenharia de características para escolher nove melhores parâmetros dentre os 41 parâmetros da abstração de fluxo. Entre várias abordagens de aprendizado de máquina, o sistema proposto funciona bem utilizando REPTree e J48. O conjunto de dados utilizado pela ferramenta foi o KDDCUP99, que pode ser considerado um conjunto de dados antigo para um sistema atual de detecção de intrusão. O sistema utiliza Apache Spark e Hadoop para processamento dos dados. Mesmo o sistema realizando a classificação em tempo real, o treinamento do modelo de aprendizado de máquina é feito de um modo *offline*.

No trabalho de Owezarski [75], é apresentado um algoritmo não supervisionado para classificação de tráfego ilícito. O algoritmo é chamado de chamado UNADA, acrônimo para *Unsupervised Network Anomaly Detection Algorithm*, em português Algoritmo de Detecção de Anomalias na Rede Não Supervisionada. O algoritmo é utilizado na identificação e caracterização de anomalias e ataques relacionados à segurança que ocorrem em *honeypots*. O autor também mostra como é possível, a partir de resultados de caracterização de anomalias, inferir regras de filtragem que possam servir para configurar automaticamente os roteadores, switches e/ou *firewalls* de rede. O desempenho do UNADA em termos de precisão de identificação de ataques é avaliado usando traços de tráfego de *honeypot* reunidos na rede *honeypot* da Universidade de Maryland. O autor assume que todo o tráfego na rede *honeypot* é malicioso, pois nenhum tráfego produtivo é utilizado nessa rede. O algoritmo proposto abre novas perspectivas para a realização de análises de risco na Internet, aproveitando o tráfego do *honeypot* e configurando automaticamente os mecanismos de proteção, como *firewalls* e IDS.

Stevanovic *et al.* [85] propõem um sistema de detecção baseado em fluxo que utiliza aprendizado de máquina supervisionado para identificar o tráfego de *botnets*. Os autores avaliaram oito algoritmos de aprendizado de máquina conceituados. Além disso, os autores avaliam quanto tráfego precisa ser observado por fluxo para capturar os padrões de tráfego malicioso. Os algoritmos de aprendizado de máquina utilizado foram *Naive Bayes*, *Bayesian Network Classifier*, Regressão Logística, Redes Neurais artificiais, Máquina de vetor suporte com kernel linear, C4.5 *decision tree* (C4.5), Árvore de decisão e Floresta aleatória. O sistema utilizou a ferramenta WEKA para realizar a classificação. Os autores utilizaram três conjunto de dados, ISOP para fluxos maliciosos e para não maliciosos foi utilizado LBNL e um conjunto de dados criado no laboratório da Ericsson. O sistema de detecção provou ser preciso na detecção de tráfego de *botnets*, usando o classificador

de árvore de decisão. Além disso, os experimentos mostraram que, para fornecer uma alta precisão de detecção, os fluxos de tráfego precisam ser monitorados apenas por um período de tempo limitado e um número limitado de pacotes por fluxo. O sistema proposto conseguiu uma detecção precisa do tráfego de *botnet* para apenas 10 pacotes e 60 segundos de monitoramento por fluxo. O sistema proposto é *offline*, sendo o módulo online citado como trabalho futuro.

Saied *et al.* [82] propõem um arcabouço para detectar e mitigar ataques DDoS conhecidos e desconhecidos em ambientes em tempo real. Os autores utilizaram Rede Neural Artificial para detectar ataques DDoS baseados em características específicas (padrões) que separam o tráfego de ataques DDoS de tráfego genuíno. O modelo, segundo os autores, foi treinado intensivamente com casos de ambientes reais e cenários de ataque de DDoS que são produzidos usando as ferramentas DDoS populares existentes. Foi observado que, quanto mais o modelo era treinado com padrões atualizados (mais recentes ataques DDoS conhecidos), mais aumentava as chances de detectar ataques DDoS conhecidos e desconhecidos. Isso ocorre porque o algoritmo de redes neurais aprende a partir de cenários e detecta padrões de ataques semelhantes ao que foi treinado. Os conjuntos de dados foram coletados, pré-processados e preparados para treinar o algoritmo usando o JNNS. O mecanismo de detecção funciona integrado com o Snort-AI, onde foi testado contra ataques DDoS conhecidos e desconhecidos. A solução foi avaliada por comparação com IDS e outras pesquisas acadêmicas relacionadas. A solução proposta não consegue lidar com ataques DDoS que usam cabeçalhos de pacotes criptografados e essa solução foi proposta para trabalhos futuros.

Bostani *et al.* [13] propõem um arcabouço de intrusão híbrida em tempo real que consiste em módulos de detecção de invasão baseados em especificação e baseados em anomalia para detectar dois ataques de roteamento bem conhecidos em *Internet of Things* (IoT), chamados de *sinkhole* e *selective-forwarding attacks*. Para isso, os agentes de detecção de intrusão estão localizados nos nós de roteamento e analisam o comportamento de seus *hosts* e enviam seus resultados locais para o nó raiz por meio da rede. Esse agente é baseado na arquitetura *Hadoop MapReduce*, então pode trabalhar em uma plataforma distribuída para projetar modelos de *clustering* e, consequentemente, detectar paralelamente anomalias em uma abordagem de detecção global. O método proposto toma decisões sobre comportamento suspeito usando um mecanismo de votação. A implantação do arcabouço híbrido proposto é investigada em um cenário de cidade inteligente por uma plataforma existente. O algoritmo de aprendizado de máquina não supervisionado usado foi o *Optimum-Path Forest Clustering* (OPFC), em português Agrupamento Florestal de Melhor Caminho. Segundo os autores, as principais diferenças entre o método proposto e outros métodos mencionados são que a estrutura proposta detecta a intrusão sem empregar mensagens de controle adicionais e monitorar os nós. Uma desvantagem do arcabouço proposto é que ele só identifica dois ataques específicos que são: *sinkhole* e *selective-forwarding attacks*.

Wang *et al.* [97] propõem um algoritmo de detecção de ataques de rede eficiente chamado *Seed-Expanding* (SE), que detecta ataques antes que eles danifiquem o sistema. O SE emprega o esquema de agrupamento de tráfego de rede de expansão de duas sementes, em inglês *Two-Seed-Expanding network traffic clustering*), cujos *clusters* analisam o tráfego em diferentes fases de ataque. Primeiro, é feito o pré-processamento do tráfego de redes, incluindo a construção do fluxo de rede, alterando as características de valor contínuo em características nominais, adotando o método de discretização e transformando-os em recursos binários. Em seguida, com base nesses recursos, o SE calcula um peso para cada fluxo e seleciona iterativamente as sementes para expandir até que todos os fluxos sejam divididos em *clusters*. Para investigar a eficácia da abordagem proposta, os autores realizaram extensas análises experimentais. Os resultados da experiência mostraram que o pré-processamento melhora muito o desempenho de agrupamento, e o algoritmo SE é melhor do que K-Means e outros tipos de semente-expansão no agrupamento de fluxos de ataque. Esses resultados de *cluster* podem ser usados posteriormente na detecção de ataques. Para a avaliação do algoritmo desenvolvido, foi utilizado o conjunto de dados DARPA2000 e o algoritmo se mostrou melhor do que os outros algoritmos testados, incluindo o K-means.

Nogueira *et al.* [74] propõem uma abordagem de detecção de *botnet* baseada na identificação de padrões de tráfego para classificá-lo como lícita ou ilícita. O sistema utiliza uma rede neural artificial para realizar a classificação. Após a fase de identificação, o sistema gera alarmes para o administrador do sistema, que poderá acionar as ações de segurança mais adequadas, como bloquear os endereços IP correspondentes, colocando-os sob uma análise mais profunda ou atuando sobre algum segmento de rede suspeito. Alguns testes de desempenho foram realizados no sistema proposto e os resultados obtidos mostraram, segundo os autores, que o sistema é estável e rápido e que a abordagem de detecção é eficiente, pois proporciona altas taxas de detecção com baixa sobrecarga computacional. Contudo, o artigo carece de métricas de avaliação frequentemente utilizados para avaliar modelos de aprendizado de máquina como barras de precisão, sensibilidade, acurácia e especificidade, curva ROC, etc. O sistema chamado de BoNeSSy possui uma interface gráfica e funciona como um IDS em uma rede de computadores.

Capítulo 3

Redes Definidas por Software

As redes de computadores e a Internet estão se tornando cada vez mais indispensáveis para a sociedade moderna. Com esse crescimento exponencial algumas redes possuem características diferentes, necessitando de protocolos específicos. A criação de novos protocolos no modelo tradicional de redes de computadores é inviável, já que os protocolos teriam que ser compilados em todos os dispositivos dessa rede para possíveis testes, o que poderia causar indisponibilidade em caso de falhas em novos protocolos. Portanto, esses fatores criam barreiras para novas ideias, pois muitas propostas de pesquisadores dessa área acabam não sendo testadas em ambientes reais. Muitos pesquisadores consideram que a infraestrutura de rede está "ossificada", não podendo ser modificada [69].

Assim, para permitir a experimentação de novas propostas para redes de computadores em uma escala menor, surgiram as redes definidas por *software* (SDN). A SDN consiste em separar o plano de dados do plano de controle, viabilizando a criação de novas ideias e protocolos em uma rede de produção. Nessa arquitetura, a rede ganha um elemento, o controlador SDN. Nele, reside o plano de controle de todos os elementos da rede, de forma que ele possui uma visão global da rede e é o responsável pelas principais decisões dessa rede. O controlador SDN, também conhecido como sistema operacional de rede, é o software que oferece uma interface de programação simples e uma visão topológica da rede, o que simplifica as tarefas da engenharia de rede [28]. Além de possibilitar a programação da rede, o controlador facilita o gerenciamento da rede. Nas redes tradicionais, os administradores tinham que gerenciar as configurações de todos os dispositivos da rede, já nas SDNs, as configurações estão mantidas no controlador da rede, facilitando,assim, a gerência de configuração. Isso permite que os administradores de rede alterem arbitrariamente tabelas de roteamento em dispositivos de roteamento de rede. Ele também permite uma camada extra de controle sobre os dados da rede, já que o administrador pode atribuir prioridades altas/baixas a determinados pacotes de dados ou permitir/bloquear determinados pacotes que trafegam pela rede [41].

O controlador SDN possui duas interfaces de comunicação, a *Northbound Interfaces*

(NBI) e *Southbound Interfaces* (SBI). A NBI é responsável pela comunicação de aplicações de alto nível com o controlador, assim outras aplicações podem se comunicar com o controlador permitindo integração com outros sistemas. Já a SBI permite a comunicações dos componentes de rede com o controlador. O protocolo SBI mais utilizado em redes definidas por *software* é o OpenFlow [51].

3.1 O Protocolo OpenFlow

O OpenFlow, proposto pela Universidade de Stanford e normatizado pela *Open Networking Foundation* (ONF), é a tecnologia de redes definidas por software mais utilizada [2, 3, 51]. De fato, o OpenFlow foi a primeira padronização de interface de comunicação entre o plano de dados e o plano de controle, em um esforço conjunto entre universidades, centros de pesquisa e indústrias de equipamentos de rede. Essa interface permitiu quebrar o paradigma de que equipamentos de rede devem ser monolíticos e fechados, como um *mainframe* [3]. O OpenFlow se baseia no conceito de sistema operacional de rede em que o plano de controle, por meio das primitivas do plano de dados da SDN, poderia conhecer e controlar todos os dispositivos da rede e suas interconexões físicas.

O OpenFlow tem como objetivo ser flexível para atender aos seguintes requisitos [2]:

- Possibilidade de uso em implementação de baixo custo e de alto desempenho;
- Capacidade de suportar uma ampla gama de pesquisas científicas;
- Garantia de isolamento entre o tráfego experimental e o tráfego de produção;
- Permitir aos fabricantes não exporem o projeto de suas plataformas.

O OpenFlow permite o acesso direto e a manipulação do plano de dados, representados por *switches* e roteadores, por meio de uma interface padronizada e um protocolo de comunicação. Nessa arquitetura, assume-se a existência de um controlador logicamente centralizado que é capaz de acessar todos os equipamentos da rede por meio de um canal seguro. É importante ressaltar que o OpenFlow transforma o equipamento de rede, seja ele *switch* ou roteador, em uma caixa configurável ou "*switch* generalizado"que depende do controlador para definir seu comportamento, não executando nenhuma ação além daquelas programadas. Assim, funções básicas da rede, como a execução do protocolo *Address Resolution Protocol* (ARP), a separação lógica da rede em *Virtual Local Area Network* (VLAN)s ou a execução de algoritmos de roteamento, deixam de ser configuradas em cada equipamento, e passam ao domínio do controlador que enviará as regras que implementarão estas funções. Isso traz alguma complexidade à programação da rede, mas, por outro lado, garante maior flexibilidade e a possibilidade do desenvolvimento de novos protocolos e funcionalidades. Além disso, o OpenFlow também introduz o conceito de

encaminhamento multicamadas. Em redes tradicionais existe o isolamento explícito entre camadas. O encaminhamento em camada de enlace, por exemplo, se faz com base no endereço físico (*Media Access Control* (MAC)) de destino, enquanto que o roteamento ocorre com base no endereço de rede (*Internet Protocol* (IP)) de destino. Em uma rede OpenFlow, o encaminhamento dos dados pode ser feito considerando diferentes campos de diferentes camadas simultaneamente. Com isso, se introduz uma flexibilidade maior na rede, além de se permitir inovação na forma de conduzir o tráfego. Por essas razões, em uma SDN baseada em OpenFlow, todos os equipamentos são chamados de *switches* OpenFlow, independente de funcionarem como *switches* ou roteadores.

Em termos de funcionalidades disponibilizadas, o OpenFlow é análogo ao conjunto de instruções de uma CPU [3]. As instruções de uma CPU definem como um sistema de computação pode utilizar a CPU, independente de sua arquitetura interna. Assim CPUs de fabricantes diferentes podem rodar o mesmo programa se implementam o mesmo conjunto de instruções. Da mesma forma, o protocolo OpenFlow especifica as primitivas básicas que podem ser usadas por aplicações externas para programar os equipamentos da rede, sem que haja a necessidade de se conhecer o fabricante ou a implementação da lógica interna do equipamento.

Outro conceito utilizado pelo OpenFlow é o de processamento de fluxos. Um fluxo de dados é definido com base em conjunto de campos do cabeçalho. Os *switches* OpenFlow encaminham fluxos com base em regras pré-definidas ou definidas sob demanda pelo controlador da rede, as quais estabelecem um determinado conjunto de comportamentos que devem ser executados a cada novo pacote de dados relacionado àquele fluxo de dados. Isso permite ajustar a granularidade do controle da rede de acordo com as especificidades de cada tipo de fluxo. Além disso, o encaminhamento de fluxos também pode ser configurado para considerar padrões de uso da rede, a ocorrência de falhas, tipos de aplicações em uso, entre outros. Isso permite uma resposta em tempo real da rede a diferentes padrões de aplicações e sessões. Uma rede OpenFlow pode ser integrada com relativa facilidade a uma infraestrutura de rede não SDN, permitindo uma migração gradual da infraestrutura existente [6].

3.1.1 Arquitetura OpenFlow

A Figura 3.1 apresenta a descrição da arquitetura de uma rede OpenFlow. O plano de dados é composto por *switches* OpenFlow, os quais se comunicam com um controlador centralizado. A comunicação entre *switches* OpenFlow e o controlador se dá por meio de um canal seguro construído utilizando *Secure Socket Layer* (SSL), através do qual são trocadas mensagens de acordo com o protocolo do OpenFlow, também chamado de *wire protocol*. Utiliza-se também um protocolo de configuração e gerência chamado *OpenFlow Configuration Protocol* (OF-CONFIG), baseado no *Network Configuration* (NETCONF), que permite a configuração remota dos parâmetros básicos dos switches [1]. Assim, enquanto

o protocolo OpenFlow é utilizado para gerenciar as tabelas de encaminhamento dos *switches*, o protocolo OF-CONFIG permite determinar parâmetros básicos de configuração, como o IP do controlador da rede.

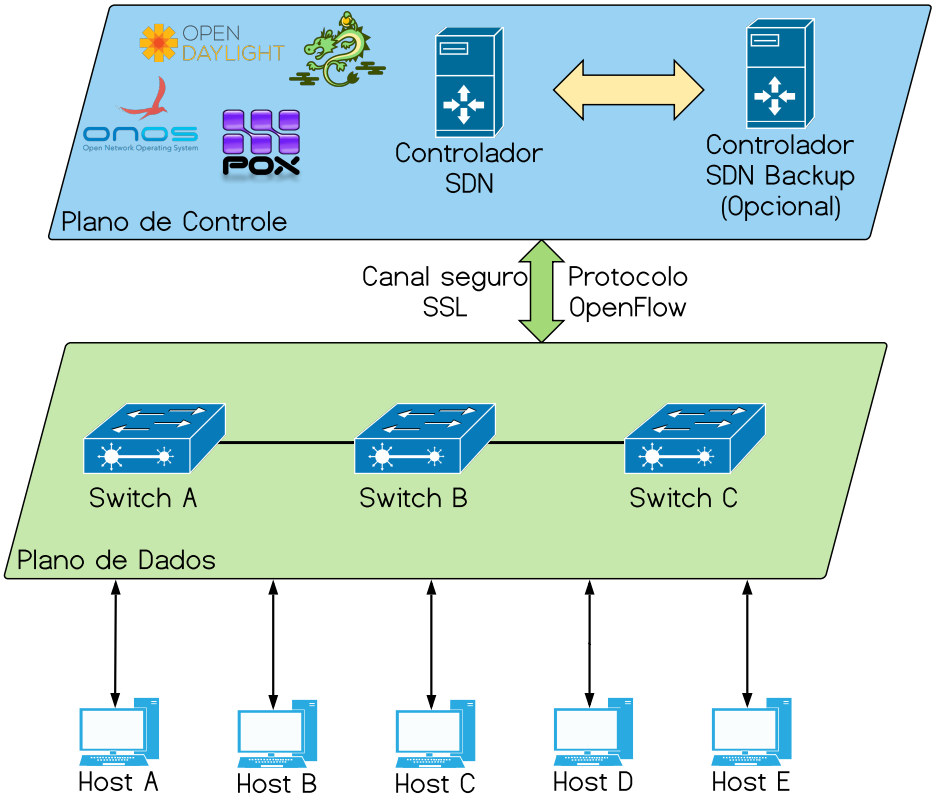
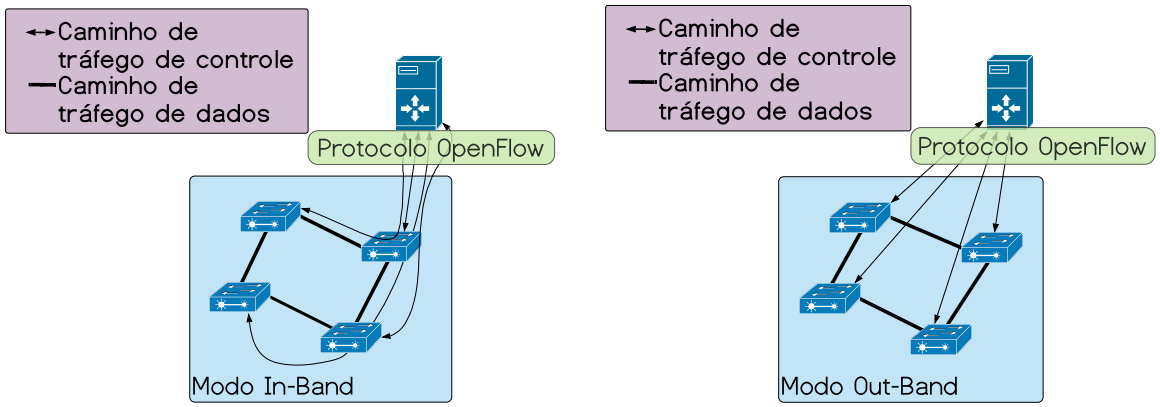


Figura 3.1: Arquitetura OpenFlow, composta pelo plano de dados e pelo plano de controle, os quais são interconectados utilizando um canal seguro e o protocolo OpenFlow.

A arquitetura OpenFlow assume a existência de um controlador centralizado, responsável por gerir os equipamentos de encaminhamento da rede. Uma vez que esse é um ponto central de falha, a plataforma prevê a utilização de controladores *backup*, os quais são acionados em caso de falha do controlador principal. Cabe observar que essa arquitetura leva à necessidade de fragmentação de redes de grande porte, de acordo com a capacidade do hardware do controlador e dos enlaces de acesso entre o controlador e os *switches* OpenFlow. Nesses casos, pode ser necessária a comunicação entre controladores.

A conexão entre o controlador e os *switches* por um canal SSL pode ser feita nos modos '*in band*' ou '*out of band*', como mostrado na Figura 3.2. No modo '*in band*', o controlador se comunica com os *switches* usando o próprio plano de dados OpenFlow. Assim, existe a necessidade de se configurar entradas de fluxo para que o *switch* chegue até o controlador. No modo '*out of band*', existe um canal externo de comunicação entre o controlador e cada *switch*.

Cabe ainda ressaltar que, por padrão, os equipamentos clientes não fazem parte do plano de dados OpenFlow. Isso significa que eles não conhecem o protocolo OpenFlow,



(a) Controle de Tráfego "in-band". (b) Controle de Tráfego "out-band".

Figura 3.2: Tipos de controle de tráfego

não encaminham pacotes e não são programados pelo controlador. Isso é importante para garantir que a rede seja transparente para os clientes finais, que farão uso de suas aplicações independentemente do uso de uma rede tradicional ou OpenFlow. Contudo, se alguma aplicação específica demandar o controle dos clientes, seria necessário garantir que eles tenham uma implementação do protocolo OpenFlow instalada e que tenham a capacidade de estabelecer uma comunicação segura com o controlador.

3.1.2 Arquitetura do *switch* OpenFlow

O plano de dados de um *switch* OpenFlow, chamado de *datapath*, é implementado por meio de tabelas de fluxo. Essas estruturas associam a descrição de um fluxo com uma ou mais ações associadas [2].

Um *switch* OpenFlow consiste em pelo menos três partes, conforme ilustrado na Figura 3.3:

1. uma tabela de fluxos, com ações associadas a cada entrada de fluxo;
2. um canal seguro que conecta o *switch* a um controlador remoto, permitindo que comandos e pacotes sejam enviados de forma segura entre um controlador e o *switch* OpenFlow; e
3. uma implementação do protocolo OpenFlow, que fornece uma maneira aberta e padrão para um controlador se comunicar com um *switch*.

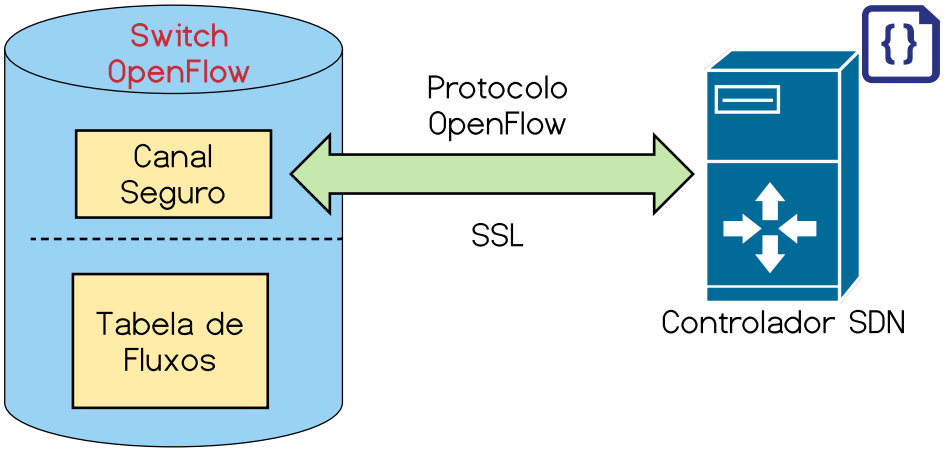


Figura 3.3: *Switch* OpenFlow, onde a tabela de fluxo é gerida por um controlador remoto através de um canal seguro

3.1.3 Tabelas do *switch* OpenFlow

As tabelas de fluxos dos *switches* OpenFlow são como as tabelas de um *switch* tradicional, associando uma descrição de fluxo à ação a ser realizada pelo *switch*. Assim, se um quadro é recebido pelo *switch* e seu cabeçalho corresponde a uma determinada regra de fluxo, uma ação é tomada, como, por exemplo, encaminhar ou descartar o pacote. Em um *switch* OpenFlow versão 1.1 ou superior, ao invés de uma única tabela, o controlador da rede pode programar um *pipeline* de tabelas, que corresponde a uma sequência de tabelas de fluxo ordenadas. Isso permite a criação de regras de encaminhamento mais complexas.

Nos *switches* tradicionais, o campo do cabeçalho de correspondência na tabela de encaminhamento é o endereço MAC de destino. Nas tabelas do OpenFlow, é possível criar regras mais elaboradas, utilizando diversos campos. Por exemplo, a versão 1.3 do protocolo OpenFlow define 13 campos para correspondência com os cabeçalhos dos quadros de entrada. A Figura 3.4 mostra um exemplo de tabela dos *switches* OpenFlow.

Cabe observar que essa tabela definida pelo OpenFlow é genérica o suficiente para fazer o encaminhamento com fluxos bem específicos, considerando, por exemplo, endereços de origem e destino e portas de aplicação utilizadas. Contudo, isso não impede o encaminhamento como na rede padrão, baseado apenas no endereço MAC de destino. Isso é possível pelo uso de máscaras nos campos que não interessam na definição do fluxo. Um campo mascarado de um quadro pode ter qualquer valor e ainda ser correspondente com aquela entrada de fluxo. Assim, regras que utilizam muitas máscaras geram uma agregação maior do que regras com todos os campos definidos, que geram uma descrição granular de fluxo.

A descrição do fluxo pelos campos de correspondência do cabeçalho é parte de uma entrada de fluxo da tabela. Essa entrada conta com os campos [4]:

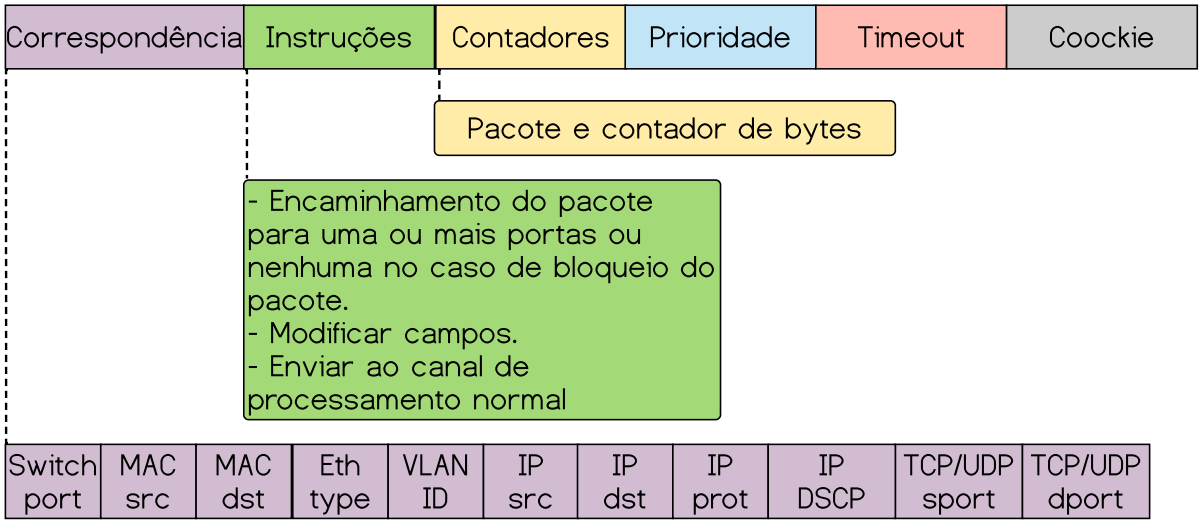


Figura 3.4: Formato simplificado de uma entrada da tabela de fluxos em um *switch* OpenFlow 1.3.

- Correspondência (*Match Fields*): são os campos de correspondência dos cabeçalhos dos pacotes. Eles consistem na porta de entrada e nos cabeçalhos de pacote e, opcionalmente, nos metadados especificados por uma tabela anterior do *pipeline* de tabelas*;

- Prioridade (*Priority*): Marca a precedência que a entrada de fluxo tem em relação às outras. Se as entradas possuírem a mesma prioridade, o *switch* pode escolher qualquer ordenação entre elas. Cabe aqui destacar que *switches* OpenFlow não fazem seleção de rota pelo tamanho da correspondência entre o prefixo do endereço IP e as rotas correspondente, como ocorreria no roteamento tradicional. Assim, mesmo que a atividade exercida pelo dispositivo seja de roteamento, o *switch* OpenFlow irá respeitar exclusivamente a prioridade e a ordem que as regras aparecem, como mostrado na Figura 3.5. Uma vez encontrada uma correspondência, nenhuma outra entrada será buscada;

- Contadores (*Counters*): São variáveis que registram dados sobre a entrada de fluxo, como o número de pacotes e bytes trafegados, o número de erros, entre diversos

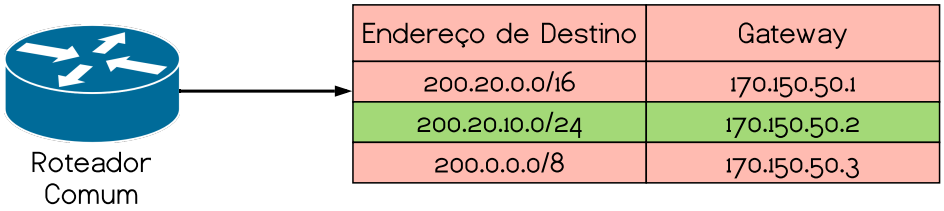
*Metadados são informações adicionadas para o processamento de um determinado fluxo, que são utilizadas exclusivamente na realização da correspondência (*match*) entre tabelas. Assim, uma determinada entrada de fluxo pode ter como ação adicionar um metadado e encaminhar o quadro para a próxima tabela. Por sua vez, essa próxima tabela pode ter uma entrada que considere o valor desse metadado para definir suas ações. Uma vez que o pacote saia do processamento interno do switch, todos os metadados são descartados.

outros. Essas informações são atualizadas sempre que há uma correspondência entre um quadro recebido e a entrada de fluxo;

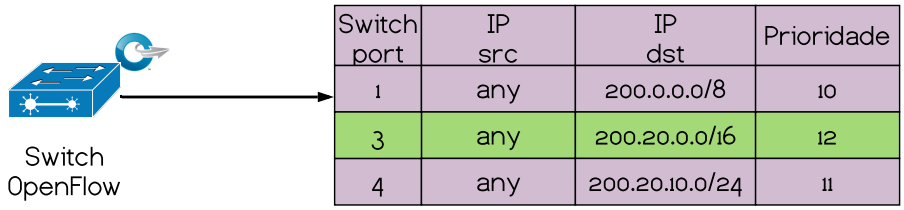
- Instruções (*Instructions*): Descrevem o que fazer com o quadro quando ele corresponder a uma regra. Essas instruções podem resultar em alterações no conteúdo do quadro e na definição sobre como encaminhar os dados (encaminhar por determinadas portas, descartar, encaminhar para outra tabela ou inserir em uma fila para provimento de qualidade de serviço).

- *Timeouts*: Variáveis que definem o tempo ocioso máximo e o tempo máximo desde a criação de uma entrada de fluxo. Caso esses limiares sejam alcançados, a entrada de fluxo expira e é automaticamente excluída do *switch*. Opcionamente, o *switch* pode comunicar ao controlador sobre a exclusão da entrada;

- *Cookie*: Valor escolhido pelo controlador. Pode ser usado pelo controlador para filtrar estatísticas de fluxo, modificação de fluxo e exclusão de fluxo. Esse campo não é usado no processamento de pacotes.



(a) O roteador irá selecionar a rota com o CIDR mais específico, independente se a rota sumarizada possui um melhor caminho.



(b) O *switch* OpenFlow irá encaminhar o pacote para o caminho com maior prioridade.

Figura 3.5: Seleção de rotas utilizando um roteador padrão e um switch OpenFlow.

Nas versões iniciais do OpenFlow, sempre que um fluxo de pacote não tinha correspondência com a tabela de fluxos, o primeiro pacote desse fluxo era encaminhado para o controlador, para que a inteligência da rede decidisse sobre o que fazer com aquele fluxo [3]. A partir da versão 1.3.0 do protocolo, essa ação deixou de ser padrão, sendo substituída pelo descarte do pacote, uma vez que a entrada de fluxo *table-miss* foi criada [4].

A lógica da entrada de fluxo *table-miss* é permitir que o *switch* diga o que fazer com os fluxos que não estão configurados até o momento. Assim, todas as tabelas de fluxo precisam ter suporte à *table-miss* para processar fluxos não definidos na tabela. Entre as ações possíveis para a *table-miss* estão enviar o pacote para o controlador, descartar o pacote ou enviar o pacote para uma tabela subsequente do *pipeline* de tabelas.

Uma entrada *table-miss* é caracterizada por ter todos os campos mascarados, ou seja, todos os campos omitidos, e por ter prioridade 0 (prioridade mínima). Isso garante que qualquer pacote que não seja classificado pelas regras da tabela de fluxo serão classificados na regra *table-miss*. A entrada *table-miss* não existe por padrão e precisa ser gerenciada pelo controlador da rede. Em linhas gerais, é uma entrada de fluxo como qualquer outra, podendo ser adicionada, removida ou expirar.

Cabe observar que, apesar do comportamento padrão do *switch* OpenFlow 1.3.0 ou superior ser descartar pacotes que não têm correspondência na tabela de fluxos, é possível mudar esse comportamento com o OF-CONFIG.

3.1.4 Instruções e ações

Cada entrada de fluxo das tabelas de fluxos deve ser associada a uma ou mais instruções. As instruções permitem criar o conjunto de ações que será aplicado sobre o fluxo durante o seu processamento.

A Tabela 3.1 traz os tipos de instruções disponíveis. É importante notar que um determinado fluxo só pode ter uma instrução de cada tipo aplicado a si em cada momento do processamento do fluxo no *switch*.

3.1.5 Conjunto de Ações (*Action set*)

O OpenFlow define um conjunto de ações em suas especificações que podem ser realizadas por um *switch* ao processar o encaminhamento de um pacote, como descrito na Tabela 3.2.

O conjunto de ações é criado/atualizado no processamento de cada uma das tabelas de fluxo. As ações do conjunto de ações são executadas mediante a ausência da instrução *Goto-Table*, o que significa que nenhuma outra tabela deve ser processada e o pacote deve ser encaminhado ou descartado pelo *switch* de acordo com as ações que estiverem definidas no conjunto de ações.

Uma entrada de fluxo pode modificar o conjunto de ações utilizando as instruções da Tabela 3.1. Por padrão o conjunto de ações inicia vazio e é preenchido/modificado ao longo do *pipeline*. Cabe observar que o conjunto de ações está associado a cada pacote individualmente e não ao fluxo como um todo.

Tabela 3.1: Lista de instruções disponíveis para configuração das tabelas do *switch* OpenFlow.

Instrução	**Descrição**	**Tipo**	**Uso**
Write-Actions	Adiciona as ações especificadas no conjunto de ações atual. Caso uma ação daquele tipo já exista, ela é sobrescrita pela ação atual	Obrigatória	*Write-Actions action(s)*
Clear-Actions	Apaga imediatamente todas as ações no conjunto de ações atual.	Opcional	*Clear-Actions*
Apply-Actions	Aplica as ações especificadas imediatamente sobre o pacote, sem causar nenhuma modificação ao conjunto de ações atuais. Em geral, essa instrução é usada para modificar o pacote entre tabelas ou para permitir executar múltiplas ações do mesmo tipo.	Opcional	*Apply-Actions action(s)*
Meter	Direciona o pacote para um meter específico. É importante notar que isso pode levar ao descarte do pacote.	Opcional	*Meter meter_id*
Write-Metadata	Escreve um metadado mascarado no campo de metadados. O uso de máscara permite que apenas parte do metadado seja modificado.	Opcional	*Write-Metadata metadata / mask*
Goto-Table	Indica a próxima tabela a ser usada no processamento do fluxo.	Obrigatória	*Goto-Table next-table-id*

3.1.6 Mensagens

O protocolo OpenFlow suporta três tipos de conjuntos de mensagens: controlador-*switch*, assíncronas e simétricas. As mensagens Controlador-*switch* são geradas pelo controlador para gerenciar e inspecionar o estado de um *switch*. As mensagens assíncronas são geradas pelo *switch* para atualizar o controlador sobre eventos da rede e mudanças no estado do *switch*. Já as mensagens simétricas podem ser geradas tanto pelo controlador quanto pelo *switch*. São enviadas sem solicitação [2].

As mensagens do tipo **controlador-*switch*** são iniciadas pelo controlador e podem ou não exigir uma resposta do *switch*, sendo elas:

- Características (*Features*): O controlador requisita as características do *switch* e o *switch* deve responder com as características suportadas;
- Configuração (*Configuration*): Usado para configurar ou solicitar configurações do *switch*;

Tabela 3.2: Lista de ações disponíveis para configuração das tabelas do *switch* OpenFlow.

Ação	**Descrição**	**Tipo**
Output	Essa ação encaminha o pacote para uma porta específica, seja ela física ou lógica.	Obrigatória
Drop	Não há uma ação explícita para descartar o pacote. Em vez disso, pacotes que não tiverem ações de *output* são descartados. Isso pode ocorrer também em um conjunto de ações vazios ou após a execução da instrução.	Obrigatória
Group	Processa o pacote através do grupo especificado. A interpretação exata depende do tipo de grupo.	Obrigatória
Set-Queue	Essa ação define a *queue id* para o pacote. Quando o pacote for encaminhado para a porta específica, a *queue id* irá determinar a qual fila o pacote pertencerá.	Opcional
Push-Tag/Pop-Tag	Ação que insere ou remove *tags* do pacote, como por exemplo *tags* de VLAN, *Multiprotocol Label Switching* (MPLS), etc.	Opcional

- Modificação de estado (*Modify-State*): Usada para adicionar, deletar e modificar a tabela de fluxos e para setar propriedades nas portas do *switch*;
- Leitura de estado (*Read-State*): Coleta para monitoramento do *switch*;
- Envio de pacote (*Send-Packet*): Utilizado para enviar pacotes por uma determinada porta do *switch*;
- Barreira (*Barrier*): Usado para garantir que as dependências foram atendidas ou para receber notificações de operações finalizadas.

Já as mensagens **assíncronas** são enviadas pelo *switch* para o controlador sem solicitação do controlador. Os *switches* enviam mensagens assíncronas para denotar uma chegada de pacote para o qual não existe entrada de fluxo, alteração de estado do *switch* ou erro. Os quatro principais tipos de mensagens assíncronas são:

- Entrada de pacotes (*Packet-In*): Utilizada quando fluxos não classificados entram no *switch*;
- Remoção de fluxo (*Flow-Removed*): Mensagem enviada quando um fluxo é removido da tabela, seja por *timeout* por ociosidade, *timeout* por tempo de existência da entrada de fluxo ou por uma mensagem de modificação da tabela de fluxos que delete a entrada em questão;
- Estado da porta (*Port-Status*): Mensagem enviada sempre que há mudanças nas configurações das portas;

- Erro (*Error*): Notificações de erros.

Por fim, as mensagens **simétricas** são enviadas sem solicitação, em qualquer direção, sendo elas:

- *Hello*: Mensagens trocadas entre o controlador e o *switch* quando uma conexão é estabelecida;
- Eco (*Echo*): Mensagens usadas para identificação de latência, largura de banda e existência de conectividade;
- Fabricante (*Vendor*): Provêem uma forma padrão para os *switches* OpenFlow oferecerem funcionalidades adicionais.

3.1.7 Processamento de pacotes pelo *switch* OpenFlow

Um pacote, ao chegar no switch, é tratado por meio de uma busca na primeira tabela de fluxos. A busca nessa tabela e em todas as demais tabelas de fluxo segue a lógica descrita na Figura 3.6.

A busca por uma entrada de fluxo é feita com base no valor da porta de entrada do pacote no *switch* e pelos campos do cabeçalho. A busca é feita por ordem de prioridade e, uma vez encontrada uma entrada compatível com o pacote, as instruções da entrada são aplicadas, atualizando o conjunto de ações para aquele pacote e os campos do cabeçalho daquele pacote. Inicialmente, quando o pacote chega ao *switch* e é levado à primeira tabela, o seu conjunto de ações está vazio. A cada processamento de uma tabela, esse conjunto de ações é atualizado, assim como os campos do pacote.

A Figura 3.7 ilustra como o pacote percorre as tabelas de fluxo de um *switch*. Começando pela tabela 0, é checado se há correspondência do pacote com as entradas na tabela de fluxo. Caso haja correspondência, o conjunto de ações (*action set*) é atualizado por meio das instruções especificadas. Caso não haja correspondência, é verificado se há uma *table-miss*. Havendo, são executadas as instruções da entrada especial. Caso não exista uma *table-miss*, o pacote é descartado.

Havendo correspondência entre o pacote e uma entrada de fluxo, após o conjunto de ações ser atualizado, é verificado se existe a instrução *Goto-Table*. Caso exista, o pacote é direcionado para a próxima tabela, mantendo sempre o conjunto de ações e os campos de cabeçalho que foram atualizados. Se não houve uma instrução *Goto-Table*, as ações do conjunto de ações são executadas.

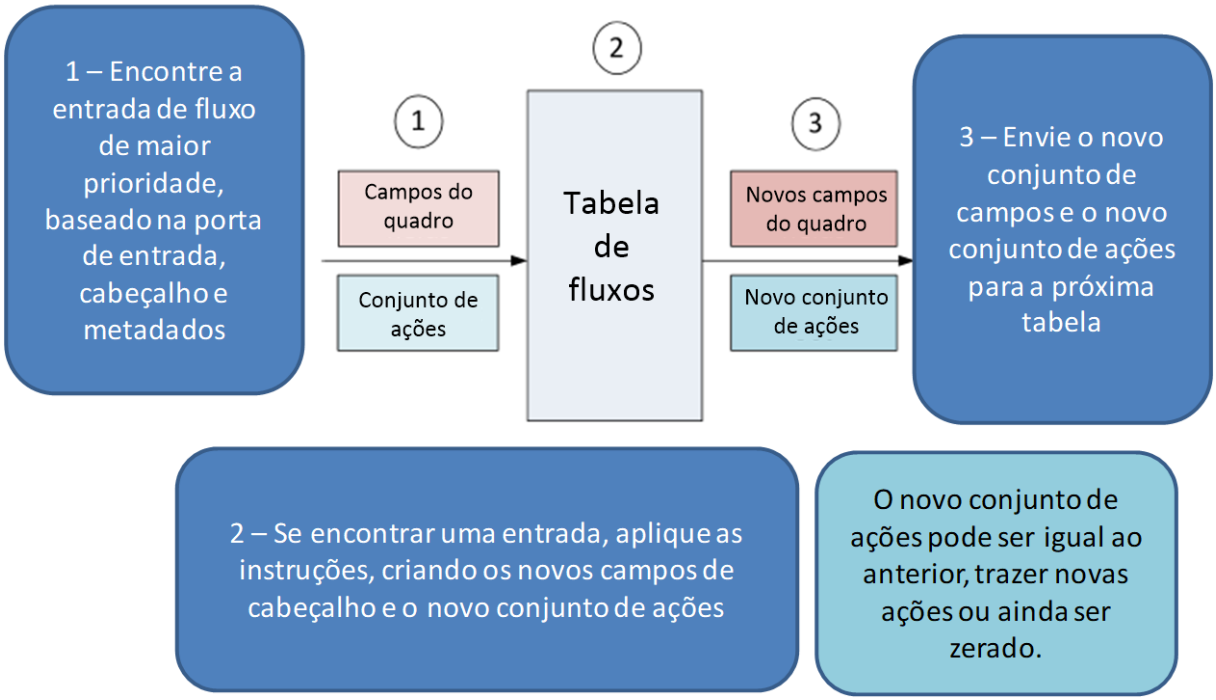


Figura 3.6: Esquema detalhando o fluxo através das tabelas de um *switch* OpenFlow.

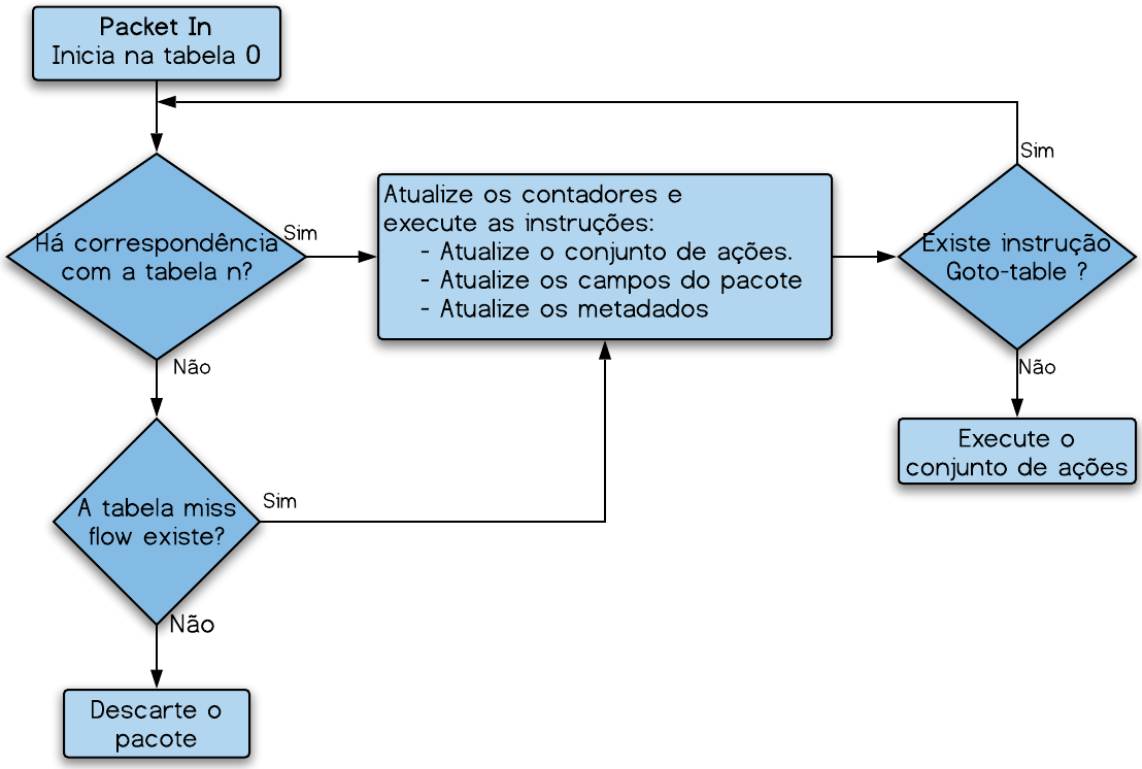


Figura 3.7: Fluxograma detalhando o fluxo através das tabelas de um *switch* OpenFlow. Adaptado de [4].

3.1.8 Vantagens e desvantagens da tecnologia

O OpenFlow é um protocolos SDN de maior destaque e que vem sendo utilizada em diversos cenários. Entre os benefícios trazidos por essa tecnologia, pode-se destacar [2, 3]:

- Gerência e controle centralizados de equipamentos de rede de fabricantes diversos;
- Possibilidade de inovação com rapidez, pois os equipamentos são configuráveis/programáveis a partir de um ponto central;
- Existência de APIs que permitem abstrair detalhes da rede;
- Possibilidade de aumento da confiabilidade da rede, pois a configuração centralizada e por uma interface padronizada é menos propensa a erros do que a configuração manual dos equipamentos;
- Possibilidade de controle mais granular da rede, com a possibilidade de aplicação de diferentes tipos de políticas, compreendendo sessão, usuário, dispositivo e aplicação;
- Possibilidade de testar novos protocolos, separando rede de produção de rede de teste;
- Facilidade na configuração da rede, pois o plano de controle encontra-se centralizado no controlador;
- Fácil implementação de QoS para aplicações específicas dentro de uma LAN, ou até mesmo na rede de um *data center*.

Cabe observar que a tecnologia também tem algumas desvantagens, tais como [80]:

- O controlador ou *cluster* de controladores acaba se tornando um ponto de falha na rede, com risco de DoS ou DDoS no controlador, comprometendo a rede;
- Possibilidade de ataque às aplicações do controlador e ao próprio controlador, caso não haja um projeto de segurança na arquitetura do software;
- Necessidade de um equipamento para controle da rede, que não é necessário nas redes tradicionais;
- Tecnologia ainda sendo amadurecida, com alta taxa de atualização de versões, as quais acabam por não ser implementadas pelos fabricantes;
- Ausência de operações transacionais, que permitiriam uma consistência nas operações de atualização dos *switches* ao longo da rede, podendo levar a inconsistências temporárias e a perda de pacotes;

Capítulo 4

Processamento de Grandes Massas de Dados

*

O rápido crescimento do número de dispositivos de rede e móveis com capacidade de comunicação sem fio contribuiu significativamente para um aumento ainda maior no volume de dados disponíveis para serem transformados em informação útil. As grandes massas de dados, muito conhecido pelo termo em inglês *Big Data*, são dados estruturados e não estruturados que são tão grande, em quantidade e tamanho, que é difícil processar usando técnicas tradicionais de banco de dados relacionais. Na maioria dos cenários empresariais, o volume de dados é muito grande ou se move muito rápido ou excede a capacidade de processamento atual. As grandes massas de dados têm o potencial de ajudar as empresas a melhorar as operações e tomar decisões mais rápidas e inteligentes. Os dados são coletados de várias fontes, incluindo e-mails, dispositivos móveis, aplicativos, bancos de dados, servidores e outros meios. Esses dados, quando capturados, formatados, manipulados, armazenados e analisados, podem ajudar uma empresa a obter *insights* úteis para aumentar as receitas, obter ou reter clientes e melhorar as operações.

Então, o processamento de grandes massas de dados, ou *big data*, é intrinsecamente relacionado com as técnicas de *analytics*, pois ambos buscam coletar inteligência de meta dados isolados. No entanto, existem três diferenças principais [68]:

- **Volume**: São esperados até 2020 50,1 bilhões de dispositivos com acesso à Internet e esse número tende a crescer exponencialmente. Mais dados cruzam a Internet a cada segundo do que foram armazenados em toda a Internet há apenas 20 anos. Isso dá às empresas a oportunidade de trabalhar com muitos petabytes de dados em um único conjunto de dados, e não apenas na Internet. Um petabyte é um

*Este capítulo é adaptado de "Análise de Dados em Redes Sem Fio de Grande Porte: Processamento em Fluxo em Tempo Real, Tendências e Desafios". Dianne S. V. Medeiros, Helio N. C. Neto, Martin Andreoni Lopez, Luiz Claudio S. Magalhães, Edelberto F. Silva, Alex B. Vieira, Natalia C. Fernandes, Diogo M. F. Mattos. Minicursos do SBRC2019

quatrilhão de bytes, ou o equivalente a cerca de 20 milhões de arquivos de armários. Um exabyte é mil vezes esse valor, ou um bilhão de gigabytes. Uma das tecnologias muito utilizadas para armazenamento, com segurança, de toda essa massa de dados de forma distribuída é o *Hadoop Distributed File System* (HDFS).

- **Velocidade**: Para muitas aplicações, a velocidade da criação de dados é ainda mais importante que o volume. Informações em tempo real ou quase em tempo real possibilitam que uma empresa seja muito mais ágil que seus concorrentes. Transformar dados brutos em informações úteis de uma maneira rápida pode fornecer uma vantagem competitiva óbvia para uma empresa. No caso do problema de segurança da informação, obviamente, quanto mais rápido os dados são coletados e analisados, mais rápido um ameaça será identificada. Com isso, essa ameaça causará menos danos a infraestrutura de um determinado ambiente.

- **Variedade**: As grandes massas de dados assumem a forma de mensagens, atualizações e imagens postadas nas redes sociais; leituras de sensores; fluxos de rede; Sinais de GPS de telefones celulares e muito mais. Muitas das fontes mais importantes das grandes massas de dados são relativamente novas. As enormes quantidades de informação das redes sociais, por exemplo, são novas comparadas com a Internet. O mesmo vale para *smartphones* e outros dispositivos móveis que agora fornecem enormes fluxos de dados vinculados as pessoas, atividades e locais. Como esses dispositivos são onipresentes, é fácil esquecer que tudo isso é muito atual. Assim, os bancos de dados estruturados que armazenaram a maioria das informações corporativas até recentemente são inadequados para armazenar e processar grandes volumes de dados. Ao mesmo tempo, os custos cada vez mais baixos de todos os elementos de armazenamento, memória, processamento, largura de banda e assim por diante, significam que as abordagens de *data-intensive* anteriormente caras estão se tornando rapidamente econômicas.

Surgem, então, diversas oportunidades para a indústria e a pesquisa produzirem inteligência a partir do grande volume de dados coletados. No entanto, esse mesmo aumento trouxe diversos desafios computacionais, já que a capacidade de processamento de uma única máquina não acompanhou o crescimento do volume de dados. Com isso, surge a necessidade de um sistema distribuído de processamento com diversos nós [103]. Nesse contexto, inúmeros modelos de programação de aglomerados computacionais (*clusters*) diferentes surgiram visando o tratamento de diversas cargas de trabalho [60, 46, 24, 103].

4.1 Análise de Grandes Massas de Dados

A análise de grandes massas de dados é o processo muitas vezes complexo de examinar conjuntos de dados grandes e variados para obter informações, incluindo padrões ocultos,

correlações desconhecidas, tendências de mercado e preferências do cliente que podem ajudar as organizações a tomar decisões de negócios informadas.O problema que essa dissertação procura solucionar é a análise em tempo real de todos os pacotes de saída de rede. O tráfego de saíde de uma rede possui características de dados infinitos, pois os pacotes são encaminhados a todo instante. A informação desejada é se o fluxo analisado é de mineração de criptomoedas ou um fluxo normal de rede.

Como pode-se ver na Figura 4.1, a análise dos dados pode ocorrer em lote ou em fluxo. Contudo, previamente à submissão dos dados para análise, é necessário prepará-los. Dessa forma, uma etapa essencial antes da análise é o pré-processamento dos dados.

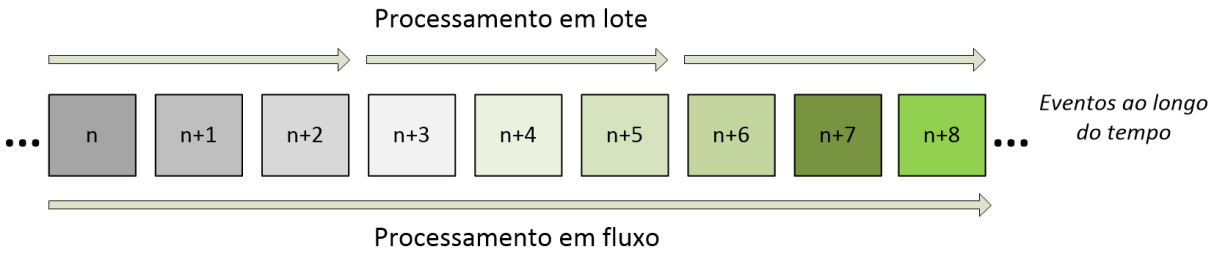


Figura 4.1: Representação do processamento de dados em lote e em fluxo; em lote onde os dados armazenados em disco são agrupados temporalmente para posteriormente serem processados; e em fluxo, no qual os dados em memória são processados assim que recebidos, gerando resposta em tempo real.

4.1.1 Análise de Dados em Lotes (*batch*)

O processamento de dados em lotes é uma forma eficiente de processar grandes volumes de dados quando as transações são coletadas em um período espaçado de tempo. Assim, os dados são coletados, armazenados, processados e, por fim, são gerados os resultados do processamento em lote. A arquitetura e implementação das bases de dados tradicionais, utilizadas no processamento em lotes, não foram projetadas para a inserção e leitura rápida e contínua de dados [71]. Em processamento de dados em lote, a informação em todo momento é coletada de uma determinada base de dados, por exemplo o HDFS, processada e posteriormente armazenada novamente. Contudo, o armazenamento em disco é um processo lento comparado as memórias de armazenamento dinâmico e volátil. Porém, isso não gera problemas, pois as técnicas para processamento em lotes não possuem fortes requisitos com relação à vazão de resultados de processamento. A análise de dados em lote é uma maneira eficiente de processamento de grandes massas de dados que estão armazenados em diversas fontes, como arquivos, banco de dados, E-mails, etc.

4.1.2 Análise de Dados em Fluxo (*stream*)

Aplicações que demandam baixo tempo de resposta, sejam elas realizadas por um operador ou automaticamente, necessitam combinar a captura de dados em fluxo com o uso de

técnicas de inferência e correlação. Devido à alta taxa de chegada de novos dados em fluxo e elevado volume de dados em uma rede de grande porte, os algoritmos de processamento tradicionais não conseguem, em tempo hábil, percorrer os dados repetidamente de forma iterativa. Para esses cenários, algoritmos de uma única passagem que utilizam pequenas quantidades de memória são necessários, caracterizando a análise de dados em fluxo e tempo real. Essa análise parte do princípio que a massa de dados não é estática e esses dados chegam constantemente, formando um fluxo de dados, como mostrado na Figura 4.1. Exemplos desse tipo de dados que são produzidos constantemente incluem *logs* de servidores *Web*, *logs* de aplicativos, dados de sensoriamento, mudanças de estado em bases de dados transacionais e fluxos de rede [16]. Nesse sentido, o uso do processamento em lotes tradicional torna-se inviável, já que o tempo das respostas do processamento só têm validade se respeitarem uma janela de tempo muito curta. Estratégias que fazem o agrupamento de dados por horas, dias ou semanas, para posteriormente processar os dados, geram um atraso intrínseco que pode tornar algumas aplicações específicas inviáveis. Ferramentas para coletas de dados, gerenciadores de fluxo de trabalho e escalonadores funcionam criando um *pipeline* contínuo de lotes [16].

Embora muitas das informações de interesse de organizações com grandes bases de dados possam ser obtidas com o processamento em lote, algumas aplicações específicas necessitam de um resultado em tempo real da análise dos dados coletados. Isso permite que a organização tome uma atitude imediata em situações que um tempo muito curto pode ser suficiente para causar problemas de diferentes naturezas. Por exemplo, a realização de uma análise para detecção de intrusão demanda uma resposta rápida, para que a intrusão não cause grandes problemas a uma máquina ou à rede. Assim, o principal objetivo do processamento de fluxos em tempo real é conseguir extrair informações para a tomada de decisão em uma janela de tempo muito curta. Ao se realizar uma análise de dados em fluxo e tempo real, os recursos de processamento e memória passam a ser um potencial gargalo, especialmente no contexto de *Big Data*. Os dados chegam em grande volume, podendo apresentar rajadas, e os resultados devem ser apresentados em tempo real ou quase real. Assim, o uso de memória secundária, em geral, não é possível nesses cenários [31], pois o tempo de escrita e de leitura é alto e incompatível com as restrições temporais da aplicação. O processamento em fluxo e tempo real utiliza o esquema "compute primeiro, armazene depois" (*compute-first, store-second*). A necessidade de resposta muito rápida para um grande volume de dados usualmente leva a necessidade do uso de sistemas distribuídos para o processamento da massa de dados.

Outras características também são importantes no processamento em fluxo, quando comparado ao processamento em lote [8], tais como, os elementos chegam em linha e devem ser processados tão logo sejam recebidos; o sistema de processamento não tem controle sobre a ordem com que os elementos chegam; os fluxos não têm restrições quanto a tamanho ou duração; e elementos processados são usualmente descartados ou arquivados e, portanto, não são acessados novamente. Os fluxos de dados podem ser representados

através de diferentes modelos.

Existem diversos requisitos que devem ser preenchidos pelas plataformas de processamento de fluxo distribuído, alguns deles são [86]: a capacidade de processar dados em linha, sem a necessidade de armazenamento prévio para efetuar as operações; a latência deve ser baixa; e não deve utilizar operações de armazenamento, como escrita e leitura em disco, que acrescentam atrasos inaceitáveis no processamento. O sistema deve ter alta disponibilidade e ter uma política de recuperação de falhas. Em aplicações de baixa latência, a recuperação deve ser rápida e eficiente, proporcionando garantias de processamento. As plataformas de processamento de fluxo devem ser resilientes contra falhas ou imperfeições, como por exemplo atrasos, perda de dados ou amostras fora de ordem, que são comuns no processamento distribuído de fluxo em aglomerados computacionais (*clusters*). O sistema deve ser independente, ou seja, não deve necessitar de instruções externas. A capacidade de processar grandes massas de dados por segundo com baixa latência, na escala de microssegundos, é essencial. Para alcançar esse desempenho, as plataformas devem minimizar a sobrecarga de comunicação entre os processos distribuídos. Os fluxos de dados podem ser representados através de diferentes modelos.

O Modelo de série temporal. Uma série temporal é um conjunto de observações de dados feitas com intervalos de tempo constantes, ao longo de uma janela de tempo. Dada uma observação de uma série temporal, ela pode ser dividida em três componentes básicos, sendo eles: tendência, relacionado a uma visão de longo prazo; sazonalidade, relacionado a movimentos sistemáticos relativos ao calendário; e irregularidade, que são flutuações de curto prazo não sistemáticas. Usualmente, uma análise de dados realiza um ajuste sazonal, identificando e removendo influências sistemáticas e relacionadas ao calendário.

O Modelo caixa registradora. Para definir esse modelo, assume-se que o sinal de entrada é representado por um fluxo $a_1, a_2, ...$ com chegada sequencial e um sinal A, definido pela função $A[j] : [1 \dots N] \rightarrow R$. As amostras a_i de entrada são incrementos para o sinal $A[j]$. Por exemplo, seja $a_i = (j, I_i)$, $I_i > 0$ e $A[j]_i = A_{i-1} + I_i$. Nesse caso, A é um fluxo no modelo caixa registradora, em que $A[j]_i$ é o estado do sinal depois da chegada da i-ésima amostra [52]. Esse é um modelo de fluxo entre os mais populares, representando, por exemplo, o tempo total de acesso de cada usuário em uma rede após múltiplos acessos, em que j representa cada usuário, I_i é o tempo de cada um de seus acessos e $A[j]_i$ representa o tempo total de acesso do usuário j até o momento i. Outro exemplo constitui o monitoramento de IPs que acessam um ponto de acesso, em que cada IP recebe um valor de j diferente e $A[j]_i$ representa o total de acessos daquele IP.

O Modelo de catraca. Esse modelo é mais genérico que os anteriores, mas é semelhante a caixa registradora. Contudo, no modelo de catraca, I_i pode assumir valores positivos ou negativos. Esse é o modelo mais apropriado para estudar situações amplamente dinâmicas, em que elementos podem ser inseridos ou retirados [52].

As plataformas de processamento de fluxo apresentam uma evolução de três gerações [8].

As plataformas de primeira geração foram baseadas nos sistemas de banco de dados, que avaliam as regras expressas como pares de condição/ação quando novos eventos chegam. Alguns exemplos de plataformas de primeira geração são Starburst [99], Postgres [87] e NiagaraCQ [18]. As plataformas de segunda geração visam estender a linguagem de consulta estruturada SQL para processar fluxos, utilizando as semelhanças entre as tecnologias [8]. STanford stREam datA Manager (STREAM), criado em maio de 2003 [9], é considerado um dos primeiros sistemas de gerenciamento de fluxo de dados de uso geral. O projeto Aurora [17] foi lançado em 2002 em uma colaboração com a Universidade Brandeis, a Universidade de Brown e o MIT, sendo um mecanismo único e centralizado de processamento de fluxo. A última versão do Aurora foi chamada de Borealis [5], que possuía melhorias como processamento distribuído e tolerância a falha. O projeto Medusa [11] utiliza a distribuição do Borealis para criar um sistema de processamento de fluxo federado. Os sistemas Borealis e Medusa tornaram-se obsoletos em 2008. As ferramentas de terceira geração surgiram para atender a necessidade de processar grandes volumes de dados e em alta velocidade de empresas *online* como Amazon, Facebook, Google, Twitter, etc. O foco principal passa a ser o processamento distribuído escalável de fluxos de dados em aglomerados computacionais [8]. O Google propõe o MapReduce [48] para o processamento paralelo escalável de grandes volumes de dados em aglomerados. A ideia principal é espalhar-processar-combinar diferentes tarefas em paralelo em servidores de um aglomerado computacionais de forma escalável. A plataforma Hadoop [49] é a implementação de código aberto do MapReduce. No entanto, devido à alta latência que o MapReduce produz, alguns outros projetos foram propostos para realizar a análise de fluxo de dados em tempo real. O projeto Spark foi baseado no Hadoop MapReduce, porém as operações são executadas em memória principal, enquanto que no Hadoop eram executadas em disco. Mesmo o Spark sendo uma plataforma de processamento em lote, ele possui uma biblioteca chamada Spark Streaming, que permite o processamento em fluxo em microlotes. As plataformas de código aberto Storm e Flink são propostas para processamento de fluxo.

4.2 Ferramentas para Processamento de Grandes Massas de Dados

No MineCap, todos os pacotes na saída da rede são processados, a fim de identificar os fluxos de mineração de criptomoeda. O MineCap utiliza o processamento de dados em fluxo devido ao grande número de pacotes de saída da rede, que combina características de alto volume de dados e alta velocidade de geração. No entanto, esse grande volume de dados também traz diversos desafios computacionais, já que a capacidade de processamento de uma única máquina não acompanhou o crescimento do volume de dados. Com isso, surge a necessidade de um sistema distribuído de processamento [103]. Nesse contexto, inúmeros modelos de programação de aglomerados computacionais (*clusters*) surgiram visando o tratamento de diversas cargas de trabalho [61, 47, 25]. Ferramentas de processamento

de dados em lote são utilizadas para trabalhar com grandes massas de dados estáticos. A ferramenta mais utilizada de processamento de dados em lote é o Hadoop [25]. Para análise de dados em redes é necessária uma ferramenta de processamento de dados em fluxo, pois os fluxos de rede são dinâmicos. Existem diversas propostas de plataformas para processamento em fluxo como Apache Spark Streaming [103], Apache Storm [45] e Apache Flink [16]. O Apache Storm é bastante utilizado, mas o Flink possui um desempenho melhor quanto ao processamento em fluxo e o Spark possui um sistema de recuperação de falhas superior [58, 19].

4.2.1 Apache Hadoop

O Apache Hadoop permite desenvolver aplicações distribuídas e paralelas, decompondo um trabalho grande em tarefas menores e um conjunto de dados massivo em partições menores, de modo que cada tarefa processa uma partição diferente em paralelo. As principais abstrações são: 1. Tarefas MAP que processam as partições de um conjunto de dados usando pares (`chave/valor`) para gerar um conjunto de resultados intermediários e 2. REDUCE, que são tarefas que mesclam todos os valores intermediários associados à mesma chave intermediária. O Hadoop usa o HDFS, que é uma implementação do *Google File System* (GFS) para armazenar dados. O HDFS é um sistema de arquivos distribuídos projetado para ser executado em hardware comum, dando suporte de tolerância a falhas em hardware de baixo custo [49].

O Arcabouço Hadoop possui dois componentes principais, o HDFS que é responsável pela manipulação e armazenamento dos dados nos nós do aglomerado computacional e o MapReduce que é responsável pelo processamento dos dados [8]. A Figura 4.2 ilustra os componentes do HDFS e MapReduce.

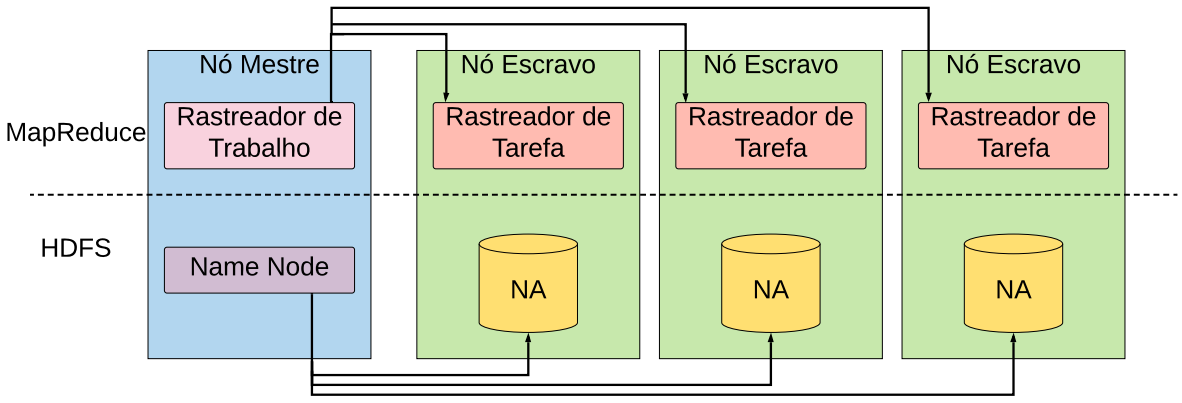


Figura 4.2: Arquitetura com os dois principais componentes do Apache Hadoop, onde NA é um Nó de Armazenamento.

O HDFS é um sistema de arquivos escalável e distribuído, que também é um sistema de arquivo distribuído. Sistemas de arquivo distribuídos são necessários, caso a quantidade de dados a serem armazenados ultrapasse a capacidade de um único *host*. Por conta disso,

toda a complexidade proveniente do ambiente de rede pode contribuir com falhas, o que faz com que sistemas de arquivos de rede sejam mais complexos do que sistemas de arquivos comuns. O HDFS armazena todos os arquivos em blocos. O tamanho do bloco padrão é 64MB, enquanto que os blocos de armazenamento comum são usualmente de 512KB. Todos os arquivos no HDFS possuem múltiplas réplicas, o que auxilia o processamento em paralelo. Os aglomerados computacionais do HDFS possuem dois tipos de nós. O primeiro um "*Name Node*", que faz parte do nó mestre, e múltiplos nós de armazenamento, que fazem parte dos nós escravos. O *Name Node* é responsável por administrar os dados no sistema de arquivos. Ele gerencia todos os arquivos e diretórios. *Name Nodes* possuem o mapeamento entre arquivos e os blocos nos quais estes estão armazenados. Já os nós de armazenamento armazenam os dados em forma de blocos. Os *Name Nodes* mantêm uma tabela de gerência mantendo informações sobre em que nós cada dado está salvo. Assim, o *Name Node* acaba se tornando o principal ponto de falha, pois sem ele os dados não podem ser acessados [8]. Para abordar tal questão, o Hadoop salva cópias das tabelas criadas pelo *Name Node* em outros nós do aglomerado computacional.

O MapReduce enxerga uma tarefa computacional como consistindo de duas fases, a fase de mapeamento (*Map*) e a fase de redução (*Reduce*), que são executadas nessa mesma sequência. As tarefas são executadas paralelamente no aglomerado computacional. O armazenamento necessário para essa funcionalidade é fornecido pelo HDFS.

O MapReduce é composto de dois componentes. O primeiro é o Rastreador de Trabalho, que submete a tarefa para os Rastreadores de Tarefas, que são executados nos nós escravos. O Rastreadores de Tarefas se reporta ao Rastreador de Trabalho em intervalos regulares, para que o Rastreador de Tarefas saiba que o nó não caiu. Caso o Rastreador de Tarefa não se reporte ao Rastreador de Trabalho, então, assume-se que houve uma queda nesse nó e seu trabalho é designado para outro Rastreador de Tarefa. O Rastreador de Trabalho, assim como o *Name Node* é um ponto crucial de falha [49].

4.2.2 Apache Spark

O Spark Streaming é uma ferramenta de processamento distribuído de dados em fluxo amplamente utilizada no mercado. O Spark é uma ferramenta de processamento distribuído em lote que possui extensões que dão suporte a diversas cargas de trabalho. O Spark possui extensões para SQL, aprendizado de máquina, processamento de grafos e processamento em fluxo, utilizando o conceito de microlotes (*microbatch*) [103]. Possui também os módulos YARN e Mesos para a gerência dos aglomerados computacionais (*clusters*), permitindo que o usuário foque na programação de novos aplicativos sem se preocupar com a localização dos dados ou em qual nó o processamento será efetuado.

A generalidade de propósito do Spark contribui para diversos benefícios: (i) facilidade no desenvolvimento de aplicações, pois o Spark possui uma API unificada; (ii) maior

eficiência na execução de tarefas de processamento e no armazenamento em memória, pois o Spark pode executar diversas funções sobre os mesmos dados, geralmente na memória, enquanto outros sistemas exigem a gravação de dados em armazenamento não volátil para que outros mecanismos os acessem; (iii) possibilidade de criação de novas aplicações, como *queries* interativas em gráficos e aprendizado de máquina em linha (*online*), o que não é possível em outras ferramentas. Para obter resiliência no armazenamento dos dados, o Spark pode utilizar o HDFS ou o *Simple Storage Service* (S3), em português, serviço simples de armazenamento, que são sistemas de arquivos para dividir, espalhar, replicar e gerenciar dados nos nós de um aglomerado computacional [8]. Com isso, os dados armazenados em disco serão distribuídos em diversos nós do aglomerado computacional, assim como os dados processados em memória, criando um ecossistema tolerante a falhas.

O compartilhamento de dados é a principal diferença entre o Spark e outras plataformas do paradigma *MapReduce*, tornando o Spark mais rápido que plataformas anteriores para consultas interativas e algoritmos iterativos, como aprendizado de máquina [100]. O Spark possui um paradigma de programação similar ao *MapReduce* do Hadoop, porém possui sua própria abstração de compartilhamento de dados, o Conjunto de Dados Distribuídos Resilientes, *Resilient Distributed Dataset* (RDD). O RDD é a principal estrutura de dados do Spark e é responsável por armazenar os dados em memória e distribuí-los garantindo a resiliência [8]. Para cada transformação de dados, como `map`, `filter` e `groupBy`, é criado um novo RDD. A manipulação dos RDD's é feita por meio de uma API que pode ser utilizada pelas linguagens Scala, Java, Python e R, em que os usuários podem simplesmente passar funções locais para serem executadas no aglomerado computacional.

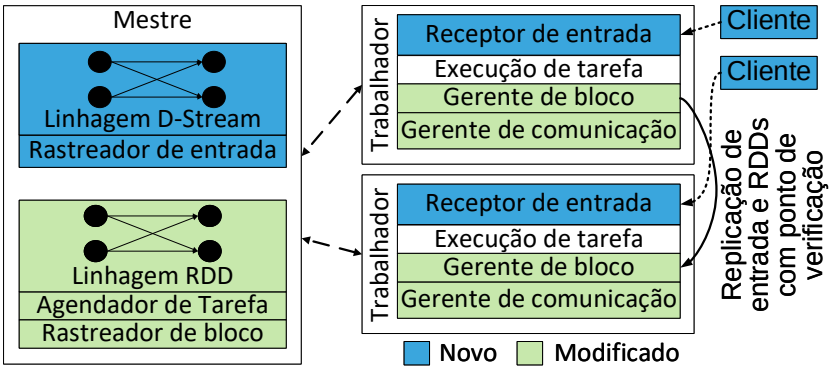
Sistemas de computação distribuída costumam prover tolerância a falha através de replicação de dados ou criando pontos de verificação (*checkpoints*). Já o Spark utiliza uma abordagem diferente chamada linhagem (*lineage*). Cada RDD grava um grafo de todas as transformações que foram utilizadas para construí-lo e retorna essas operações para uma base de dados para uma futura reconstrução no caso de perda de dados. Recuperação baseada em linhagem é significantemente mais eficiente do que replicação, obtendo um ganho de tempo, pois transferir dados pela rede é muito mais lento do que a escrita na memória RAM e no armazenamento. A recuperação é muito mais rápida do que simplesmente executar novamente o programa, pois o nó com falha geralmente possui múltiplas partições RDD's e essas partições podem ser reconstruídas em paralelo com outros nós [101]. O Apache Spark possui quatro bibliotecas de alto nível que fazem o uso da estrutura RDD para uma variedade de funções.

A **Spark SQL** implementa *queries* em linguagem de consulta estruturada SQL em banco de dados relacional. A ideia principal dessa biblioteca é ter o mesmo *layout* de banco de dados analíticos dentro dos RDD's. O Spark SQL provê uma abstração de alto nível para transformações de dados chamadas de *DataFrames*, que são abstrações comuns para dados tubulares em Python e R, com métodos programáticos para filtros e agregações [10]. A **GraphX** provê uma interface de computação de operações sobre grafos similar ao

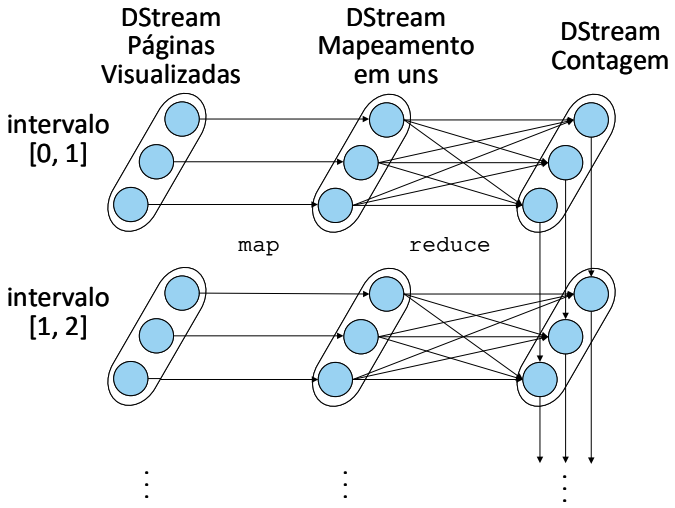
Pregel [65] e GraphLab [62] e implementa a otimização de posicionamento desses sistemas por meio de sua opção de função de particionamento para o RDD [33]. A **MLlib** é a biblioteca responsável pelas funções de aprendizado de máquina, possuindo algoritmos de classificação, regressão, agrupamentos, filtros colaborativos, engenharia de características, construção de *pipelines*, e outras funções. O MLlib permite que algoritmos de aprendizado de máquina operem de forma distribuída em aglomerados de computadores. O MLlib foi escrito em Scala e usa bibliotecas nativas de álgebra linear baseadas em C++. O MLlib possui APIs para Java, Scala e Python e é distribuído como parte do projeto Spark [72]. O MLlib possui uma documentação rica, descrevendo todos os métodos suportados e códigos de exemplo para cada linguagem suportada.

A quarta biblioteca de alto nível do Spark o habilita para o processamento de dados em fluxo. O **Spark Streaming** implementa um modelo de processamento de fluxo discretizado (*Discretized Streaming* - D-Streams), que viabiliza a tolerância a falhas e trata os dados retardatários introduzidos por nós mais lentos [102]. O Spark Streaming é composto por três componentes. O **Mestre** que monitora o grafo de linhagem do D-Stream e organiza as amostras para calcular novas partições RDD. Os **Nós trabalhadores**, que recebem e processam os dados, também armazenam as partições de entrada e calculam os RDDs. O **Cliente** que envia os dados para a ferramenta. O Spark Streaming modifica e adiciona vários componentes do Spark para ativar o processamento em fluxo. A Figura 4.3(a) mostra os componentes novos e modificados pelo Spark Streaming. A comunicação de rede, por exemplo, é reescrita para permitir que tarefas com entradas remotas sejam recuperadas rapidamente. Existem modificações no mecanismo de linhagem, como remoção de grafos de linhagens que já possuem ponto de verificação (*checkpoint*), assim evitando que os grafos cresçam indefinidamente. Do ponto de vista arquitetural, o que diferencia o Spark Streaming de outras ferramentas que operam sobre dados em fluxo é a divisão das tarefas computacionais em instâncias determinísticas, curtas, sem estado, cada uma delas podendo ser executada em qualquer nó do aglomerado computacional, ou até mesmo em vários nós. O D-Stream faz a discretização de um fluxo de dados em pequenos lotes chamados de microlotes para que eles possam ser tratados pelas tarefas computacionais adequadas.

As aplicações de *big data* costumam utilizar duas abordagens para lidar com tolerância a falha: replicação e *upstream backup*. Na replicação, há duas cópias dos dados processados e registros de entrada são replicados para outros nós, porém somente a replicação não é suficiente. É necessário também um protocolo de sincronização para garantir que as cópias de cada operador vejam os dados herdados na mesma ordem. A replicação é custosa computacionalmente, embora a recuperação de uma falha seja rápida. Já em *upstream backup*, cada nó retém uma cópia do dado enviado desde um determinado ponto de verificação. Quando um nó falha, uma máquina em modo de espera assume o seu papel e os nós pais reproduzem as mensagens a essa máquina, que antes estava em modo de espera, para que ela mude o seu estado. Ambas as abordagens possuem desvantagens. A replicação tem um grande consumo de recursos de hardware e o *upstream backup* é



(a) Arquitetura do Spark Streaming, mostrando seus componentes. Adaptado de [102]



(b) Grafo de linhagem de um programa de contagem de visualização de páginas WEB. Adaptado de [102]

Figura 4.3: Arquitetura do Spark Streaming e grafo de linhagem de um D-Stream.

lento no processo de recuperação. Ambas as abordagens não conseguem lidar com dados retardatários. A ideia do D-Stream é computar os dados em fluxo sem estado (*stateless*) utilizando computação em lote determinística em intervalos curtos de tempo.

O D-Stream estrutura as computações da seguinte maneira: i) o estado em cada etapa temporal é totalmente determinístico ao dado de entrada, abolindo a necessidade de protocolos de sincronização e ii) a dependência entre o estado atual e o antigo são visíveis em uma fina granularidade. Isso garante ao Spark Streaming um excelente mecanismo de recuperação, similar aos de sistemas em lote e que supera a replicação e *upstream backup*. A latência do D-Stream é baixa, pois ele usa a estrutura de dados RDD, que computa os dados na memória e utiliza a abordagem de linhagem, conforme ilustrado na Figura 4.3(b). O D-Stream possui uma rápida recuperação de falhas e dados retardatários, pois utiliza o determinismo para prover um novo mecanismo de recuperação que é a recuperação paralela.

Quando um nó falha, cada nó do *cluster* trabalha para computar novamente e recuperar os RDD's do nó em falha, resultando em uma recuperação significantemente mais rápida do que as outras duas abordagens [102]. O D-Stream pode recuperar retardatários utilizando execução especulativa [25]. Cada D-Stream, assim como os RDD's, são imutáveis, assim cada transformação gera um novo D-Stream. Um D-Stream é uma sequência de conjunto de dados imutáveis e particionadas que podem ser influenciados pela transformação determinística.

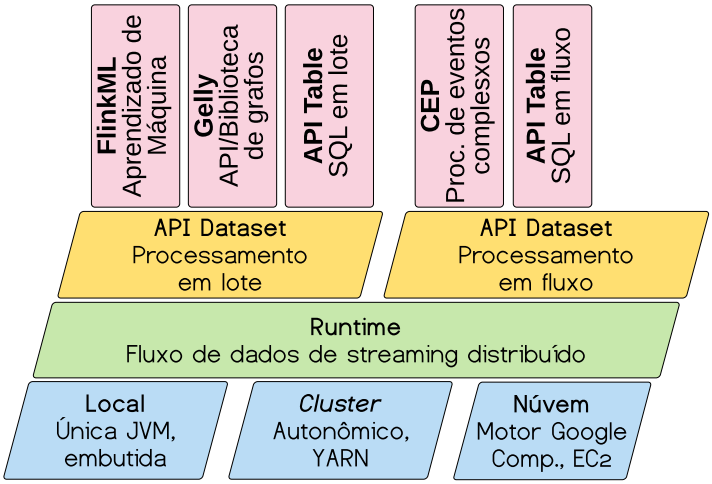
4.2.3 Apache Flink

O Apache Flink [15] foi proposto em 2009 com o nome Stratosphere [98]. Foi incubado na fundação Apache em 2014 e em dezembro do mesmo ano foi promovido a projeto *Top Level*. O Flink suporta processamento de fluxo e lote, tornando-se uma plataforma de processamento híbrida. A arquitetura do Flink pode ser distinguida em quatro camadas, Desenvolvimento, Núcleo, APIs e Bibliotecas, conforme apresentados na Figura 4.4(a).

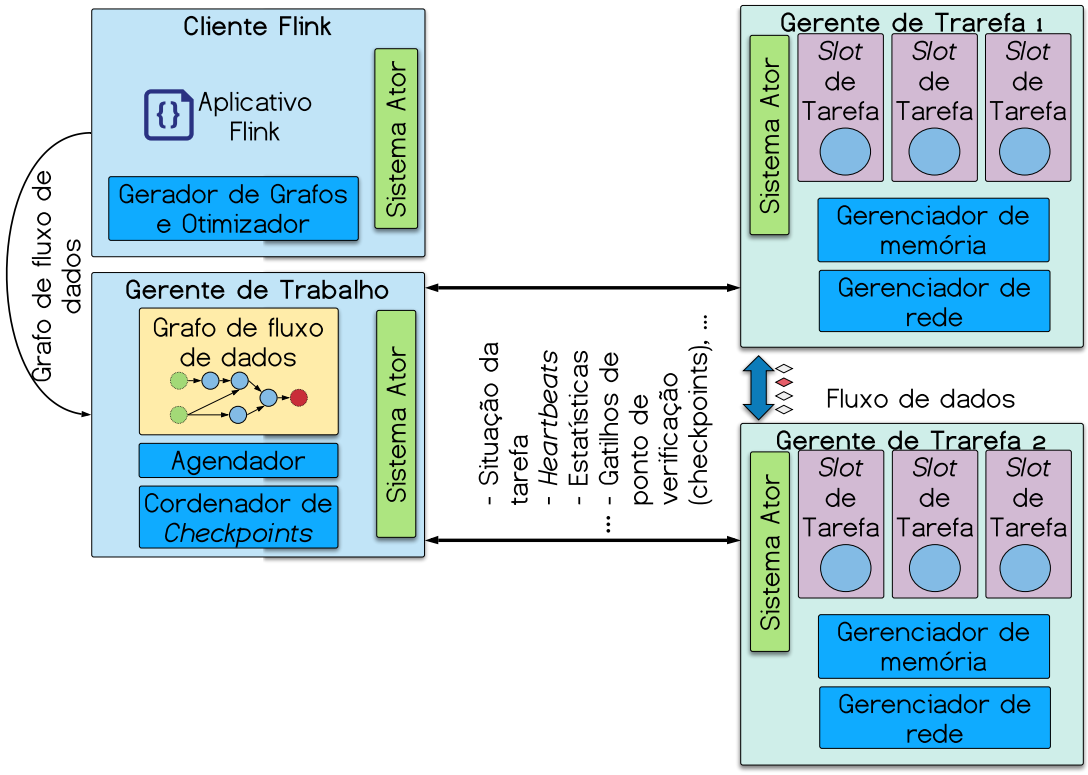
O núcleo do Flink é o motor distribuído de fluxo de dados, que executam os programas desenvolvidos. As tarefas de análise do Flink são abstraídas em grafos acíclico dirigido (GAD) formados por quatro componentes: fontes; operadores; torneiras de saída; e registros que percorrem o grafo. O grafo apresentado na Figura 4.5, consiste em: i) operadores de estado (*stateful*) e ii) fluxos de dados que representam dados produzidos por um operador e estão disponíveis para consumo pelos operadores. Como os grafos de fluxo de dados são executados de forma paralela, os operadores são paralelizados em uma ou mais instâncias, chamadas subtarefas, e fluxos são divididos em uma ou mais partições. Os operadores de estado, que podem ser sem estado, em casos especiais, implementam toda a lógica de processamento, como operações de `filter`, `map`, `hash join`, `window` etc. Os fluxos distribuem dados entre operadores de produção e consumo em vários padrões, como ponto a ponto, propagação (*broadcast*), repartição, difusão (*fan-out*) e mesclagem.

O Flink possui duas APIs bases, a API *DataSet* para processamento de dados estáticos e finitos, frequentemente associados a processamento em lote, e a API *DataStream* para processamento de dados dinâmicos e potencialmente ilimitados, frequentemente associada a processamento em fluxo. As bibliotecas do Flink são específicas para cada API e são *FlinkML* para aprendizado de máquina, *Gelly* para processamento de grafos e *Table* para processamento de operações SQL.

Um aglomerado computacional do Flink trabalha no modelo mestre-escravo conforme mostrado na Figura 4.4(b) e é composto por três tipos de processos: o cliente, o Gerente de Trabalhos e pelo menos um Gerente de Tarefas. O cliente pega o código do programa Flink, o transforma em um gráfico de fluxo de dados e o envia para o gerente de trabalho. O gerente de trabalho coordena a execução distribuída do fluxo de dados. Ele rastreia o estado e o progresso de cada operador e fluxo, programa novos operadores e coordena



(a) Arquitetura do Flink. Adaptado [15]



(b) Modelo de processamento do Flink. Adaptado [15]

Figura 4.4: Arquitetura e modo de trabalho do Flink.

os pontos de verificação (*checkpoints*) e a recuperação. Em uma configuração de alta disponibilidade, o gerente de trabalho persiste em um conjunto mínimo de metadados em cada ponto de verificação para um armazenamento tolerante a falhas, de modo que um Gerente de Trabalho em espera possa reconstruir o ponto de verificação e recuperar a execução do fluxo de dados a partir dele. O processamento de dados ocorre nos Gerentes de Tarefas. Um Gerente de Tarefas executa um ou mais operadores que produzem fluxos e

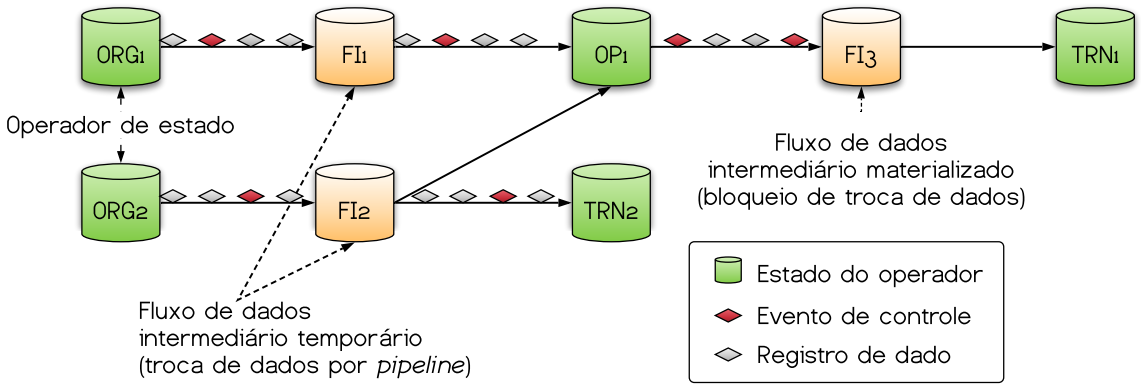


Figura 4.5: Grafo de fluxo de dados. Adaptado [15]

relatórios sobre seu status para o Gerente de Trabalho. Os Gerentes de Tarefas mantêm os *buffer* para armazenar ou materializar os fluxos e as conexões de rede para trocar os fluxos de dados entre os operadores [15]. A abstração do fluxo de dados no Flink é chamada *DataStream* e é definida como uma sequência de registros parcialmente ordenados. Parcialmente, pois não há garantia de ordem caso um elemento operador receba mais de um fluxo de dados como entrada [8].

O Flink oferece execução confiável com garantias de consistência da semântica de processamento "exatamente uma" e lida com falhas utilizando ponto de verificação e reexecução parcial. O mecanismo de verificação do Apache Flink baseia-se em captura momentânea do estado do sistema (*snapshot*) distribuídos para obter garantias de processamento exatamente uma vez. A natureza possivelmente ilimitada de um fluxo de dados torna a recomputação na recuperação impraticável, já que possivelmente meses de computação precisarão ser repetidos para uma tarefa longa. Para diminuir o tempo de recuperação, o Flink obtém uma captura de estado *snapshot* dos operadores, incluindo a posição atual dos fluxos de entrada em intervalos regulares. O mecanismo usado no Flink é chamado de Captura de estado de Barreiras Assíncronas *Asynchronous Barrier Snapshotting* (ABS) [15]. As barreiras são registros de controle injetados nos fluxos de entrada que correspondem a um tempo lógico e separam logicamente o fluxo para a parte cujos efeitos serão incluídos na captura atual e a parte que será capturada posteriormente. Um operador recebe barreiras de *upstream* e primeiro executa uma fase de alinhamento, certificando-se de que as barreiras de todas as entradas foram recebidas. Então, o operador escreve seu estado, como por exemplo, conteúdo de uma janela deslizante ou uma estrutura de dados customizada em um sistema de arquivo durável, com o HDFS por exemplo. A recuperação de falhas reverte todos os estados do operador para seus respectivos estados retirados da última captura de estado bem-sucedida e reinicia os fluxos de entrada a partir da última barreira para a qual há uma captura de estado. A quantidade máxima de recálculo necessária na recuperação é limitada à quantidade de registros de entrada entre duas barreiras consecutivas. Além disso, a recuperação parcial de uma subtarefa com falha é possível, reproduzindo registros não processados armazenados em *buffer* nas subtarefas

do fluxo ascendente.

Capítulo 5

Pré-processamento de Dados

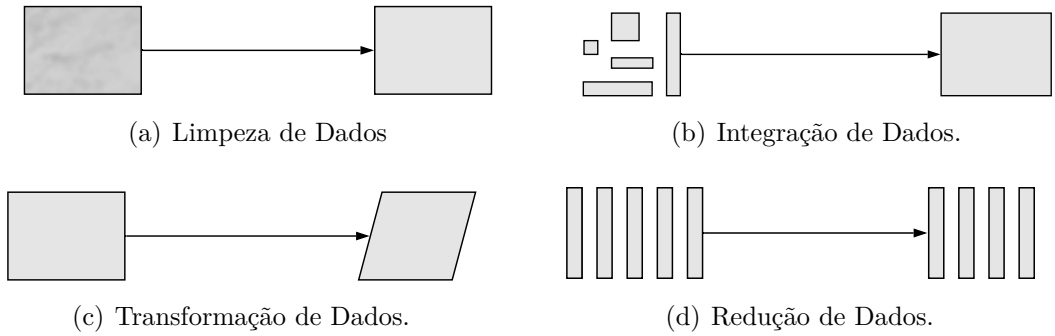
*

O pré-processamento de dados é uma fase primordial na ciência de dados. Por serem coletados de diversas fontes, os conjuntos de dados coletados podem ter diferentes níveis de qualidade em termos de ruído, redundância, consistência, etc. O pré-processamento de dados é capaz de adaptar os dados aos requisitos apresentados por cada algoritmo de aprendizado de máquina, permitindo processar dados que seriam inviáveis em sua forma primária [42]. Os dados em sua forma bruta, também são conhecidos como dados primários [93], como apresentam baixa qualidade, ao serem processados diretamente resultam em modelos de baixo desempenho e ainda utilizam um grande espaço de armazenamento. O objetivo do pré-processamento é justamente aumentar a qualidade do conjunto de dados obtido e, possivelmente, reduzir o seu volume através da limpeza de inconsistências e ruídos, eliminação da redundância e integração dos dados para fornecer uma visão unificada. Essa etapa pode ocorrer antes ou após a transmissão dos dados, sendo mais adequada a sua execução antes da transmissão, para fins de economia de banda e de espaço de armazenamento [42].

Após a realização do pré-processamento, o conjunto de dados obtido pode ser considerado confiável e adequado para a aplicação de algoritmos de mineração de dados [93]. O pré-processamento de dados é de especial importância quando há grande volume de dados coletados e redundância entre eles. Por exemplo, dois pontos de acesso distintos podem coletar dados sobre a rede em uma região onde existe sobreposição de cobertura. Os dados coletados por ambos os pontos de acesso referentes a essa região em comum são fortemente redundantes, sendo desnecessário armazená-los por completo. Além disso, caso fossem processados com a presença da redundância, as conclusões sobre a análise poderiam ser tendenciosas. O pré-processamento inclui diversas técnicas para promover a *limpeza*, a

*Este capítulo é adaptado de "Análise de Dados em Redes Sem Fio de Grande Porte: Processamento em Fluxo em Tempo Real, Tendências e Desafios". Dianne S. V. Medeiros, Helio N. C. Neto, Martin Andreoni Lopez, Luiz Claudio S. Magalhães, Edelberto F. Silva, Alex B. Vieira, Natalia C. Fernandes, Diogo M. F. Mattos. Minicursos do SBRC2019

integração, a *redução* e a *transformação* dos dados [42, 30], como ilustrado na Figura 5.1.



(a) Limpeza de Dados (b) Integração de Dados.

(c) Transformação de Dados. (d) Redução de Dados.

Figura 5.1: Técnicas de Pré-processamento de Dados. Adaptado de [30].

5.1 Limpeza de Dados

As técnicas para limpeza dos dados determinam a acurácia e a completude dos dados, consertando os problemas encontrados através da remoção ou adição de dados ao conjunto original e resolvendo inconsistências. Uma estrutura ideal para limpeza de dados consiste em cinco etapas complementares [42], que são:

- Definir e determinar tipos de erros;
- Procurar e identificar amostras com erro;
- Corrigir os erros;
- Documentar as amostras e os tipos de erros;
- Modificar os procedimentos de entrada de dados para reduzir erros futuros.

Conjuntos de dados reais frequentemente são incompletos devido a falhas no processo de coleta dos dados, limitações no processo de aquisição dos dados ou restrições de custo, que impedem o armazenamento de alguns valores que deveriam estar presentes no conjunto de dados [30]. Além disso, é necessário tratar os erros nos dados, documentando tanto as ocorrências quanto os tipos de erros para que os procedimentos possam ser modificados a fim de reduzir erros futuros [42].

Na primeira etapa da limpeza, busca-se por dados discrepantes usando diversas ferramentas comerciais de depuração de dados (*data scrubbing tools*) e de auditoria de dados (*data auditing tools*). Ferramentas de depuração de dados usam conhecimentos simples sobre o domínio dos dados para detectar erros e corrigi-los, empregando técnicas de análise (*parsing*) e de correspondência difusa (*fuzzy matching*). Já as ferramentas

de auditoria de dados descobrem regras e relações, identificando dados que violam as condições encontradas. Para tanto, empregam, por exemplo, análise estatística para identificar correlações e algoritmos de agrupamento para identificar *outliers*.

Quando se identifica a ausência de dados, a solução trivial é descartar a amostra, mas isso pode resultar na deturpação do processo de aprendizagem e no descarte de informações importantes. Outra abordagem é completar os dados manualmente, o que pode ser impraticável. Uma constante global também pode ser utilizada para completar os dados ausentes, resultando, potencialmente, em uma interpretação errada dos dados. Em vez de uma constante, um valor de tendência central para a característica como a média ou mediana pode ser utilizado. A estratégia mais utilizada é determinar o valor ausente através de modelos probabilísticos aproximados, utilizando procedimentos de máxima verossimilhança [30].

A presença de erros, ou ruído, nos dados é tratada através de duas abordagens principais, polimento de dados (*data polishing*) e filtros de ruído [53]. O resultado de ambas é semelhante, produzindo uma saída suavizada através de técnicas de regressão, análise de *outliers* e técnicas de categorização (*binning*). Uma sobrecarga computacional extra significativa pode ser adicionada dependendo da complexidade do modelo utilizado. Dessa forma, é necessário buscar o equilíbrio entre o custo adicional da limpeza dos dados e a melhoria da acurácia dos resultados obtidos [42].

5.2 Integração de Dados

As técnicas de integração de dados, ou resumo, combinam os dados provenientes de diferentes fontes, unificando a visão sobre eles e reduzindo a redundância. A eliminação de redundância também é feita através de técnicas de detecção de redundância e de compressão de dados, permitindo diminuir a sobrecarga na transmissão e no armazenamento dos dados. A redundância pode se apresentar na forma espacial, temporal, estatística ou perceptiva e está fortemente presente em dados de vídeo e imagem. Técnicas para tratamento de redundância, nesses casos, já são muito bem conhecidas, como JPEG e PNG para imagem e MPEG-2, MPEG-4, H.263 e H.264/AVC para vídeo, mas não são aplicáveis ao contexto de análise de dados em redes sem fio de grande porte. Nesse caso, técnicas de remoção de duplicatas são mais adequadas.

Cada instância pode ser composta por diversas características provenientes de fontes diferentes e que podem apresentar esquemas diferentes. Os metadados das características passam a ser importantes para evitar erros durante a integração dos esquemas. Características que podem ser derivadas de outras características ou de um conjunto de características devem ser eliminadas se puderem ser consideradas redundantes. A redundância pode ser avaliada através da análise de correlação, que utiliza o teste *qui-quadrado* (χ^2) para características nominais, e o coeficiente de correlação e covariância para características

numéricas. A duplicação de dados também deve ser observada na etapa de integração e a consistência dos dados deve ser garantida, de forma que durante a integração dos dados é necessário detectar e resolver conflitos nos valores das características [30].

Duas estratégias para integração dos dados incluem o uso de Armazém de Dados (*Data Warehouse*) e Federação de Dados (*Data Federation*) [42]. No primeiro caso, a integração passa pelas etapas de extração, transformação e carregamento dos dados. Na extração, seleciona-se os dados necessários para a análise, que são modificados na etapa de transformação através da aplicação de um conjunto de regras, convertendo os dados em um formato padrão. Por fim, os dados transformados são importados para a infraestrutura de armazenamento e geralmente são provenientes de uma única fonte [42]. No segundo caso, cria-se um banco de dados virtual para consultar e agregar dados de fontes diferentes. O banco de dados virtual contém apenas informação ou metadados sobre os dados reais e sua localização, não armazenando os dados de fato. Essas abordagens são adequadas para processamento de dados em lote, mas ineficientes para dados em fluxo.

5.3 Redução de Dados

As técnicas de redução dos dados objetivam diminuir o volume do conjunto de dados, mas de forma que o resultado produzido após o processamento seja mantido. As técnicas incluem estratégias de redução de dimensionalidade e de numerosidade. Também são utilizadas técnicas de compressão de dados para reduzir o volume de dados. A redução de dimensionalidade busca uma representação comprimida dos dados originais, focando na diminuição da quantidade de variáveis aleatórias ou características consideradas. Para tanto, são usadas as Transformadas *Wavelet*, a Análise de Componentes Principais PCA e Seleção de Características. Também podem ser empregados algoritmos genéticos para reduzir de forma ótima a dimensão do conjunto de dados [90]. Na redução de numerosidade, os dados são substituídos por representações alternativas, de formato menor, usando técnicas paramétricas ou não-paramétricas. As paramétricas, como regressão e modelos log-lineares, estimam os dados e armazenam apenas os parâmetros do modelo que os descrevem. As técnicas não-paramétricas incluem histogramas, algoritmos de agrupamento, técnicas de amostragem e agregação de cubos de dados. Por fim, as técnicas de compressão de dados empregadas podem resultar ou não em perdas. Se houver perdas, os dados reconstruídos serão apenas uma aproximação dos dados originais. No caso em que não há perdas, os dados são reconstruídos integralmente.

A *Transformada Wavelet* é uma técnica de processamento de sinais linear em que o sinal passa a ser representado por uma soma de formas de onda mais simples. A transformada *wavelet* tem como objetivo capturar tendências em funções numéricas, decompondo o sinal em um conjunto de coeficientes. Assim, o vetor de dados passa a ser representado por um vetor de coeficientes *wavelet* de mesmo comprimento. Existe uma família de

transformadas *Wavelet* que podem ser usadas no pré-processamento dos dados, sendo as mais populares Haar-2, Daubechies-4 e Daubechcies-6 [37]. Os coeficientes mais baixos podem ser descartados sem prejuízo significativo para a recuperação dos dados originais. Através do armazenamento de uma pequena fração dos coeficientes de maior intensidade, os dados originais são obtidos aplicando a transformada inversa.

O *PCA* cria um conjunto menor de variáveis que permite representar os dados originais. O PCA é considerado um método de extração de características. Esses dados podem ser descritos através de um vetor de n características ou dimensões e o PCA combina a essência das características originais usando k vetores ortogonais de dimensão n, com $k \leq n$, que melhor representem os dados a serem reduzidos. Como o espaço dimensional de projeção dos dados é menor, a dimensionalidade dos dados é reduzida. Nesse novo espaço dimensional, o PCA pode revelar relações que anteriormente não estavam claras, permitindo interpretações que normalmente seriam ignoradas.

A *Seleção de Características* remove características redundantes ou irrelevantes, mantendo um conjunto mínimo de características que resulte em uma distribuição de probabilidade próxima da distribuição original. A seleção de características reduz o risco de *overfitting* dos algoritmos de mineração de dados e reduz o espaço de busca, tornando o processo de aprendizagem mais rápido e consumindo menos memória [30]. Dados representados por um vetor de n características possuem 2^n possíveis subconjuntos, dos quais deve-se escolher uma quantidade ótima para representar os dados. Devido ao grande número de possibilidades que podem ser exploradas para chegar ao resultado ótimo, geralmente são utilizadas heurísticas que exploram um conjunto reduzido do espaço de busca. Geralmente, são usados algoritmos gulosos (*greedy*), que escolhem sempre soluções ótimas locais, como seleção passo a passo para frente (*stepwise forward selection*), eliminação passo a passo para trás (*stepwise backward elimination*), uma combinação dos dois anteriores e indução de árvores de decisão. Outra abordagem utiliza o teste de independência qui-quadrado para decidir quais características devem ser selecionadas.

5.4 Transformação de dados

A transformação uniformiza a representação dos dados e geralmente utiliza métodos para reduzir a complexidade e a dimensionalidade do conjunto de dados [93], podendo ser incluída também na etapa de redução dos dados [30]. As principais estratégias para transformar os dados são as técnicas de suavização, construção de características, agregação, normalização, discretização e geração de hierarquias de conceitos para dados nominais. A suavização remove ruído dos dados utilizando, por exemplo, técnicas de regressão e de agrupamento. A construção de características tem como objetivo criar novas características baseadas em outras para melhorar a acurácia e a compreensão de dados de grande dimensionalidade. Juntamente com a seleção de características, a construção de características integra uma

área de pesquisa em crescimento conhecida como engenharia de características (*feature engineering*). Os procedimentos de seleção de características identificam e removem o máximo de informação redundante e irrelevante possível [30], enquanto a extração e a construção combinam as características do conjunto original para obter um novo conjunto de variáveis menos redundantes. Na agregação, os dados são resumidos, resultando em uma nova representação. O resumo pode ser feito na forma de média, variância, máximo e mínimo. Por exemplo, o consumo diário de energia pode ser agregado para mostrar o consumo mensal máximo, ou anual médio. A normalização modifica a escala dos dados para se ajustarem em uma faixa de valores menor. A discretização substitui a representação de valores numéricos por intervalos numéricos ou rótulos conceituais. Por fim, a geração de hierarquias de conceitos para dados nominais cria múltiplos níveis de granularidade em que os dados podem ser encaixados.

Apesar de a área de pré-processamento de dados já ser bastante explorada, ainda existe muita atividade de pesquisa para buscar novos métodos e aprimorar os métodos existentes, de forma que sejam capazes de lidar com o volume de dados disponíveis cada vez maior e com a geração de conhecimento em tempo real. Existe na literatura trabalhos que estudam novos métodos para, por exemplo, dividir os dados entre processadores em sistemas em nuvem [36] e realizar seleção de características em dados em fluxo [59].

Capítulo 6

Aprendizado de Máquina

*

Após a preparação dos dados pelo pré-processamento, a análise de grandes massas de dados transforma os dados em conhecimento usando técnicas de aprendizado de máquina. O aprendizado de máquina permite inferir soluções para problemas que possam ser representados por um amplo conjunto de dados. As aplicações de aprendizado de máquina são projetadas para identificar e explorar padrões ocultos em dados, para descrever o resultado como um agrupamento de dados em problemas de agrupamento, para prever o resultado de eventos futuros em problemas de classificação e problemas de regressão, e para avaliar o resultado de uma sequência de amostra de dados para problemas de extração de regras [14]. Assim, de forma simplificada, é possível dividir os problemas de aprendizado de máquina em quatro categorias, agrupamentos (*clustering*), classificação, regressão e extração de regras.

Problemas de agrupamento têm como objetivo minimizar a distância entre amostras de dados com características similares em um mesmo grupo, enquanto maximizam a separação entre grupos distintos. *Problemas de classificação* caracterizam-se por mapearem as entradas em classes-alvos de saída que são representadas pelo conjunto discreto de valores de saída. A coesão dos resultados dos classificadores e dos agrupamentos utiliza frequentemente como parâmetro a Soma dos Erros Quadráticos (*Sum of Squared Errors* - SSE), Equação 6.1, que leva em consideração o número de agrupamentos, k; a quantidade de dados no i-ésimo agrupamento, n_i; o j-ésimo dado no i-ésimo agrupamento, x_{ij}; a média dos dados no i-ésimo agrupamento, c_i; e a quantidade de dados, n [93]. A Soma dos Erros Quadráticos é dada por

*Este capítulo é adaptado de "Análise de Dados em Redes Sem Fio de Grande Porte: Processamento em Fluxo em Tempo Real, Tendências e Desafios". Dianne S. V. Medeiros, Helio N. C. Neto, Martin Andreoni Lopez, Luiz Claudio S. Magalhães, Edelberto F. Silva, Alex B. Vieira, Natalia C. Fernandes, Diogo M. F. Mattos. Minicursos do SBRC2019

	Classificação positiva	Classificação negativa
Positivo (P)	Verdadeiro positivo (VP)	Falso negativo (FN)
Negativo (N)	Falso positivo (FP)	Verdadeiro negativo (VN)

Tabela 6.1: Matriz de confusão

$$SSE = \sum_{i=1}^{k} \sum_{j=1}^{n_i} D(x_{ij} - c_i) \qquad (6.1) \qquad\qquad c_i = \frac{1}{n_i} \sum_{j=1}^{n_i} x_{ij}. \qquad (6.2)$$

A métrica de distância, D, mais comum é a distância Euclidiana, definida na Equação 6.3, onde p_i e p_j são posições de dois dados diferentes. Outras distâncias podem ser usadas, como as distâncias Manhattan e Mikowski.

$$D(P - i, p_j) = (\sum_{l=1}^{d} |p_{il}, p_{jl}|^2)^{\frac{1}{2}} \qquad (6.3)$$

Métricas usadas para avaliar os resultados da classificação são a acurácia (Equação 6.4), a precisão (Equação 6.5), a revocação (Equação 6.6), a especificidade 6.7 e F-measure (Equação 6.8), que permitem descobrir os acertos (verdadeiros positivos, VP, e verdadeiros negativos, VN) e as falhas na classificação, como a quantidade de dados que não pertencem ao grupo mas são incorretamente classificadas como pertencentes (falsos positivos, FP) e a quantidade de dados que pertencem ao grupo que não são classificados como pertencente ao grupo (falsos negativos, FN). Essa interpretação pode ser sumarizada através de uma matriz de confusão do classificador, conforme indicado na Tabela 6.1. Outra métrica importante na avaliação de classificadores é a Área Abaixo da Curva da Característica de Operação do Receptor ROC, conhecida como *Area Under the Curve* (AUC), que permite verificar o compromisso entre a taxa de verdadeiros positivos e a taxa de falsos positivos. O tamanho da AUC é diretamente proporcional ao desempenho do classificador. A Figura6.1 demonstra o comportamento da curva ROC. Andreoni Lopez *et al.* [8] explicam detalhadamente cada uma dessas métricas.

$$acc = \frac{VP + VN}{P + N}. \quad (6.4) \qquad p = \frac{VP}{VP + FP}. \quad (6.5) \qquad r = \frac{VP}{VP + FN}. \quad (6.6)$$

$$e = \frac{VN}{FP + VN}. \qquad (6.7) \qquad\qquad F = \frac{2pr}{p + r}. \qquad (6.8)$$

Os *problemas de regressão* tendem a gerar modelos matemáticos que tendem a se comportar como funções multivariáveis em que os valores de entrada são mapeados em valores contínuos de saída. Já os *problemas de extração* de regras são diferentes dos demais, pois o objetivo desse problema não é inferir valores de saída para cada conjunto de entrada, mas identificar relações estatísticas entre os dados.

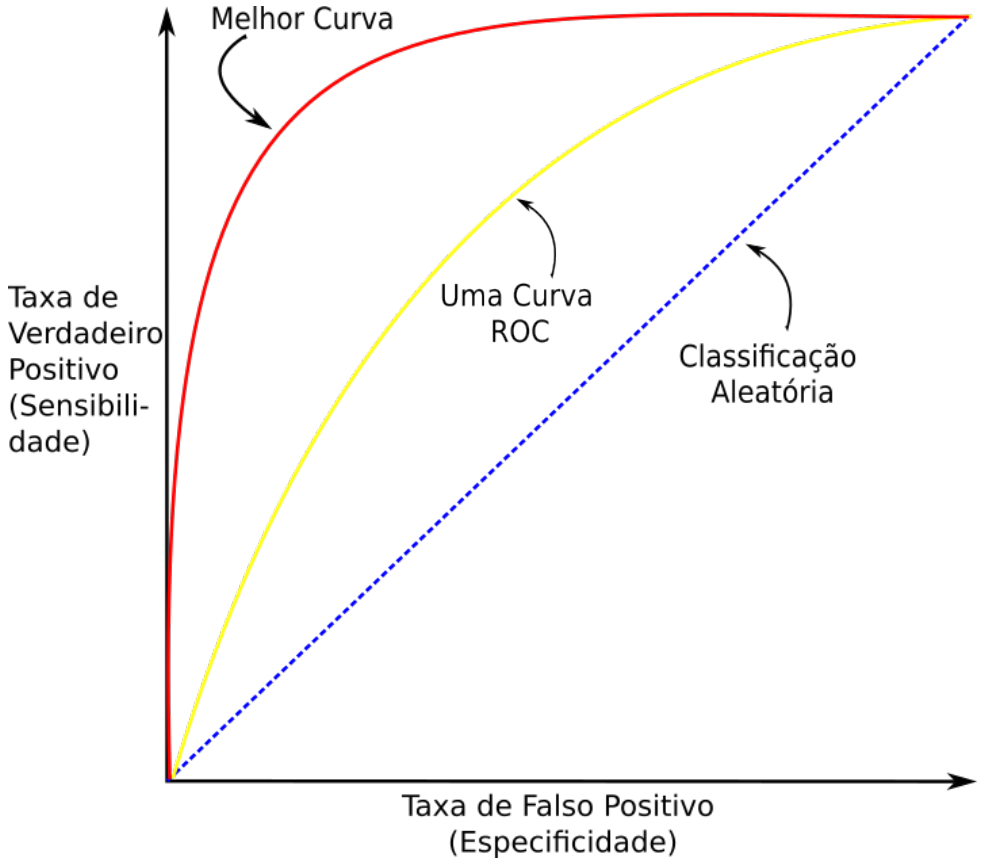


Figura 6.1: Demonstração de uma curva ROC, na qual a linha reta na diagonal representa uma classificação aleatória. Quanto mais próximo a curva estiver das bordas superior e esquerda, mais preciso será o teste. Da mesma forma, quanto mais próximo a curva estiver da diagonal, menos preciso é o teste. Um teste perfeito iria direto de zero até o canto superior esquerdo e depois direto pela horizontal.

6.1 Os Paradigmas de Aprendizado de Máquina

Os principais paradigmas de aprendizagem de máquina são supervisionados, não-supervisionados, semi-supervisionados e aprendizado por reforço (*reinforcement learning*). A definição do paradigma de aprendizagem a ser usado em uma aplicação de aprendizado de máquina determina como os dados devem ser coletados. O estabelecimento da verdade básica é responsável pela inserção de rótulos dos dados e a engenharia de características é responsável por estabelecer as características dos dados e quais delas devem ser exploradas na aplicação. A Figura 6.2 mostra os paradigmas de aprendizado de máquina e suas respectivas categorias.

A Figura 6.3 mostra o processamento em fluxo e em lote com processos que representam os quatro paradigmas de aprendizado de máquina.

Figura 6.2: Mapa mental dos principais paradigmas de aprendizado de máquina e suas categorias.

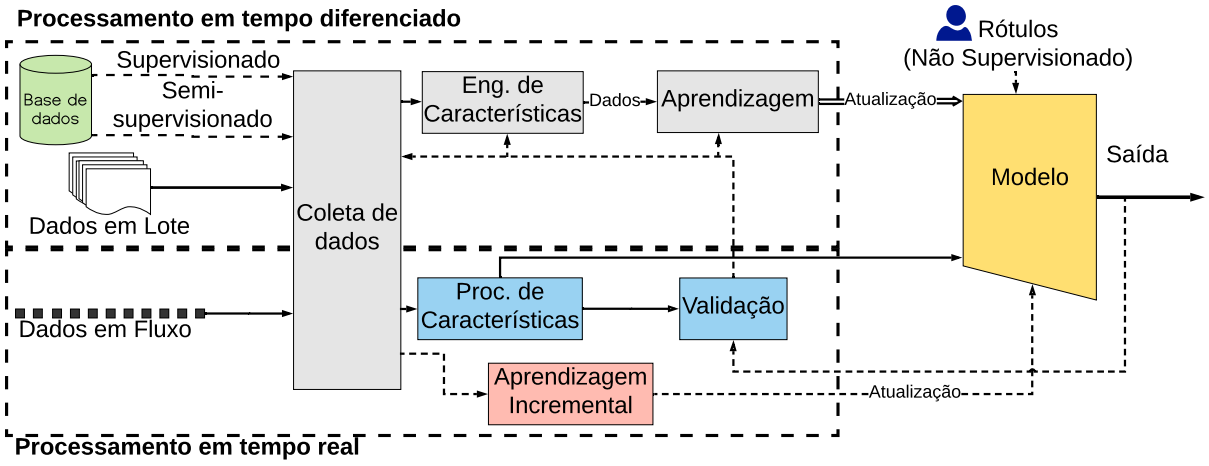


Figura 6.3: Representação dos quatro paradigmas de aprendizado de máquina em um sistema de processamento de dados em lote e em fluxo. A característica contínua e ilimitada dos dados viabiliza o aprendizado incremental, em que o modelo aprende com os dados entrantes. Já no processamento em lote, o treinamento do modelo é somente realizado com bases históricas armazenadas.

6.1.1 Aprendizado Supervisionado

O aprendizado supervisionado baseia-se em um conjunto de dados rotulados, chamado de conjunto de treinamento, para criar o modelo de classificação ou regressão dos dados. Esse paradigma de aprendizagem requer a existência *a priori* de um conjunto de dados rotulados para a criação do modelo. Por sua vez, o paradigma de *aprendizado semi-supervisionado* suporta o uso de conjuntos de treinamento com rótulos faltantes ou incompletos. A Figura 6.4 mostra um diagrama de blocos ilustrando o paradigma de aprendizado supervisionado.

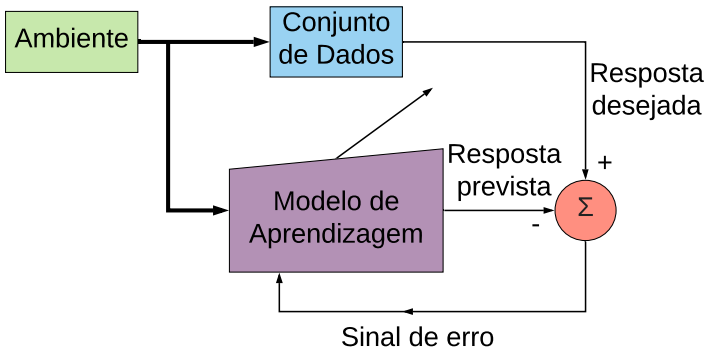
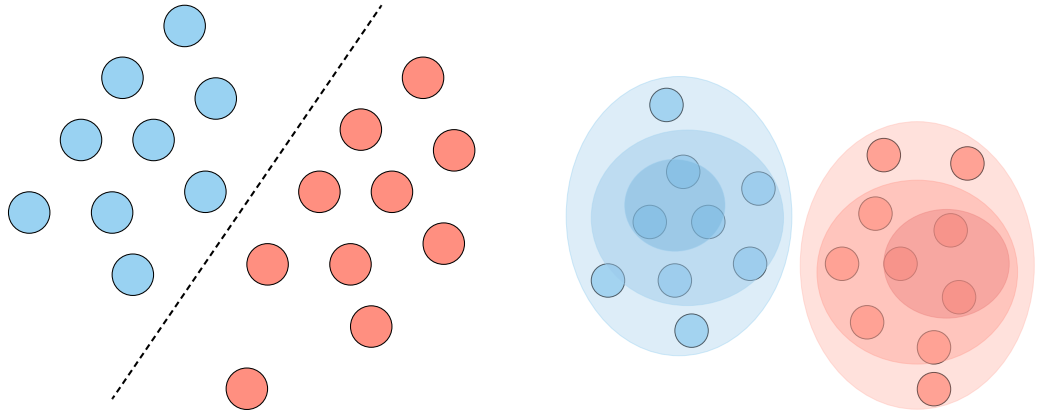


Figura 6.4: Diagrama em blocos do paradigma de aprendizagem supervisionada. Um conjunto de dados é coletado do ambiente com amostras já rotuladas, cada amostra é enviada para o modelo de aprendizagem para previsão. As respostas previstas são comparadas com as respostas desejadas, o erro é calculado para que os parâmetros sejam ajustados. Adaptado de [39]

O conjunto de dados é extraído do ambiente. Esse conjunto possui amostras com características de entrada e sua respectiva saída desejada. Suponha que uma amostra do conjunto de dados é exposta ao modelo de aprendizado de máquina. Então, essa amostra é representada por um vetor de treinamento. Assim, essa amostra do conjunto de dados contém as características de entrada que serão apresentados ao modelo e sua respectiva saída desejada. Os parâmetros do modelo são ajustados sob a influência combinada do vetor de treinamento e do sinal de erro [39]. O sinal de erro é definido como a diferença entre a resposta desejada e a resposta prevista pelo modelo de aprendizado de máquina. O ajuste é feito iterativamente com o objetivo de fazer o modelo de aprendizado de máquina emular o conjunto de dados, assim o conhecimento contido no conjunto de dados é transferido para o modelo de aprendizado de máquina.

Cada conjunto de dados é um vetor de características $\{x^{(1)}, ..., x^{(n)}\}$ relacionados a um vetor de resultados $\{y^{(1)}, ..., y^{(n)}\}$. A função do modelo de aprendizado de máquina é prever o vetor y baseado no vetor x. As previsões podem retornar valores contínuos, para problemas de regressão, ou valores discretos, para problemas de classificação. Os modelos de aprendizado de máquina podem ser discriminativos ou generativos. No modelo discriminativo tenta-se estimar diretamente $P(x|y)$ criando uma fronteira de

decisão, enquanto o modelo generativo tenta estimar para daí deduzir $P(x|y)$, gerando a probabilidade da distribuição dos dados. A Figura 6.5 ilustra a diferença entre modelos discriminativos e modelos generativos.



(a) Modelo Discriminativo (b) Modelo Generativo

Figura 6.5: Diferença entre os modelos de aprendizado de máquina supervisionado discriminativo e generativo.

A hipótese h_θ é o modelo escolhido. Para uma determinada amostra $x^{(i)}$ o resultado da predição do modelo é $h_\theta(x^{(i)})$. Utiliza-se uma função de perda para calcular o erro entre o valor real e o valor previsto. A função de perda é definida como $L : (z, y) \in \mathbb{R} \times Y \mapsto L(z, y)$ que recebe como entradas o valor z previsto correspondente ao valor real y e retorna o quão diferente eles são. A Tabela 6.2 contém as principais funções de perda utilizadas.

Erro quadrático mínimo	Perda logística	Perda de Hinge	Entropia cruzada
$\frac{1}{2}(y-z)^2$	$\log(1+\exp(-yz))$	$\max(0, 1-yz)$	$-[y\log(z)+(1-y)\log(1-z)]$

Tabela 6.2: Funções de perda.

A função de custo J é frequentemente utilizada para avaliar a performance de um modelo e é definida usando a função de perda L. A função de custo é definida como $J(\theta) = \sum_{i=1}^{n} L(h_\theta(x^{(i)}), y^{(i)})$. Para minimizar a função de custo usualmente utiliza-se a função de otimização Descida Gradiente (*Gradient Descent*). A função Descida Gradiente é definida como $\theta \leftarrow \theta - \alpha\nabla J(\theta)$, onde $\alpha \in \mathbb{R}$ é a taxa de aprendizado, que é a regra de atualização para a descida gradiente. A Descida Gradiente Estocástica (*Stochastic Gradient Descent* (SGD)), atualiza o parâmetro baseado em cada amostra do conjunto de treinamento e a decida do gradiente em lote atualiza os parâmetros baseado em todo o conjunto de treinamento.

Os principais algoritmos de aprendizado supervisionado são: regressão linear, regressão logística, máquina de vetor suporte, *Naive Bayes*, modelos baseados em árvores e k-vizinhos mais próximos (*k-Nearest Neighbors* (KNN)).

6.1.2 Aprendizado Não-Supervisionado

Já o *aprendizado não-supervisionado* busca padrões nos dados de treinamento e, portanto, não requer a existência prévia de dados rotulados. O aprendizado não-supervisionado é adequado para problemas de agrupamentos (*clustering*), em que se busca um padrão de agrupamento dos dados para maximizar a distância entre grupos, enquanto se minimiza a distância entre dados dentro de um mesmo grupo. O aprendizado não supervisionado também pode ser utilizado na redução de dimensão de um conjunto de dados, onde encontra-se direções de maximização de variância em que se projetam os dados.

Em problemas de agrupamento, o objetivo do aprendizado não-supervisionado é encontrar padrões em dados sem rótulo $\{x^{(1)}, ..., x^{(m)}\}$. O algoritmo de maximização de expectativa EM [73] fornece um método eficiente para estimar o parâmetro θ através da probabilidade máxima estimada ao construir repetidamente uma fronteira inferior na probabilidade (E-step) e otimizar essa fronteira inferior (M-step). A Figura 6.6 ilustra os passos do algoritmo maximização de expectativa.

- **E-step**: Realiza uma avaliação da probabilidade $Q_i(z^{(i)})$ na qual cada ponto $x^{(i)}$ veio de um grupo particular $z^{(i)}$, como na equação $Q_i(z^{(i)}) = P(z_i|x_i;\theta)$.
- **M-step**: Usa as probabilidades do passo anterior $Q_i(z^{(i)})$ como grupo específico de pesos nos pontos $x^{(i)}$ para, separadamente, estimar a qual grupo cada ponto pertence, como em $\theta_i = \arg\max_\theta \sum_i \int_{z^{(i)}} Q_i(z^{(i)}) \log\left(\frac{P(x^{(i)},z^{(i)};\theta))}{Q_i(z^{(i)})}\right) dz^{(i)}$.

No algoritmo k-médias (*k-means*) [38], procura-se agrupar um conjunto de dados baseados em uma centroide. O valor k determina a quantidade de centroides existentes. Então, as centroides são iniciadas $c^{(k)}$, que são os grupos de pontos de dados k e μ_j o centro do grupo j. As centroides dos grupos são aleatoriamente iniciadas como $\mu_1, \mu_2, ..., \mu_k \in \mathbb{R}$. O algoritmo k-médias repete os seguintes passos até a convergência: $c^{(k)} = \arg\min_j \left\|x^k - \mu_j\right\|^2$ e $\mu_j = \frac{\sum_{k=1}^m 1_{\{c^{(k)}=j\}} x^{(k)}}{\sum_{k=1}^m 1_{\{c^{(k)}=j\}}}$. Para verificar se o algoritmo está convergindo utiliza-se a função de distorção (*distortion function*). A Função de distorção é definida como $J(c,\mu) = \sum_{k=1}^m \left\|x^{(i)} - \mu_{c^{(k)}}\right\|^2$. O paradigma de aprendizado não-supervisionado também é utilizado para realizar redução de dimensões, sendo os algoritmos PCA e *Independent Component Analysis* (ICA) exemplos de algoritmos.

6.1.3 Aprendizado por Reforço

Por fim, o paradigma de *aprendizado por reforço* consiste em um processo iterativo, em que agentes aprendem efetivamente novos conhecimentos na ausência de qualquer modelo explícito do sistema, com pouco ou nenhum conhecimento do ambiente [91]. A ideia central do aprendizado por reforço é que o aprendizado de um agente é baseado em exemplos

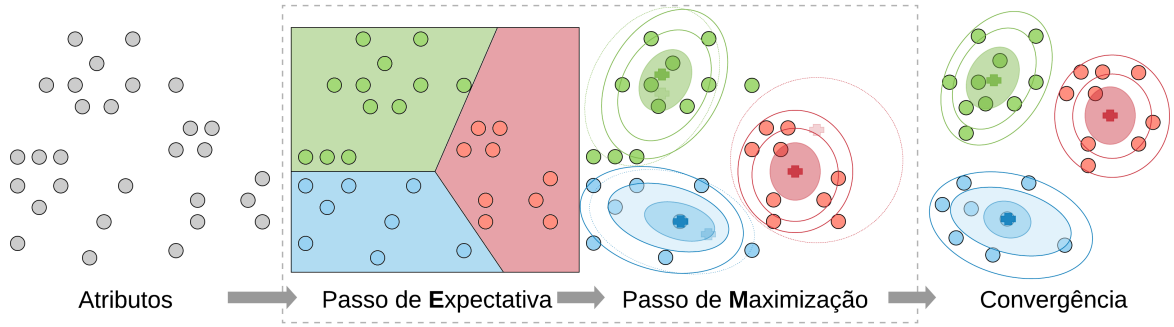


Figura 6.6: Passos do algoritmo maximização de expectativa. O que é calculado no primeiro passo (E) são os parâmetros fixos dependentes de dados da função Q. Uma vez conhecidos os parâmetros de Q, é totalmente determinado e é maximizado no segundo passo (M) de um algoritmo de maximização de expectativa.

do conjunto de treinamento, que interage com o mundo externo e aprende com reforços providos pelo ambiente. Os reforços providos podem ser recompensas ou penalidades. Assim, o conjunto de treinamento consiste de pares de amostras de dados e reforços, sejam recompensas ou penalidades. A retroalimentação do ambiente impulsiona o agente para a melhor sequência de ações. O aprendizado por reforço é adequado para problemas de tomada de decisão, planejamento e programação [91].

Capítulo 7

A proposta Super Aprendizado Incremental

A técnica de super aprendizado incremental, proposta nesta dissertação, é a combinação das técnicas *super learner* [94] e aprendizado incremental [27]. Na proposta, algoritmos de aprendizado de máquina são usados como modelos candidatos que alimentam o super modelo. Os modelos candidatos são treinados e realizam as previsões que são entregues como entrada para o super modelo. Então, posteriormente, o super modelo é treinado parcialmente utilizando o aprendizado incremental.

7.1 *Super Learner*

A técnica do *super learner* [94] consiste em treinar um modelo no qual as amostras de entrada são a saída de outros modelos, semelhante a um sistema hierárquico. Van der Laan *et al.* [94] foram os primeiros a usar esta técnica, utilizando a validação cruzada, para propor um novo método de previsão criado a partir da combinação ponderada de diversos modelos candidatos. O primeiro passo do *super learner* proposto por van der Laan *et al.* é treinar n modelos utilizando todo o conjunto de dados. No segundo passo, é feita a divisão do conjunto de dados original em V blocos e, depois, cada modelo candidato é treinado com um desses blocos, chamados de blocos de validação. Logo, é feita a previsão dos blocos de dados de validação correspondente a cada bloco de treinamento. Na sequência, são feitos ajustes do resultado observado dos resultados previstos de cada modelo candidato e, por fim, é feito o treinamento do super modelo com os resultados previstos pelos modelos candidatos. Pode-se avaliar o super modelo comparando com as previsões de cada modelo candidato, feito no primeiro passo, com a saída do super modelo. A Figura 7.1 mostra o diagrama da técnica original do *super learner*.

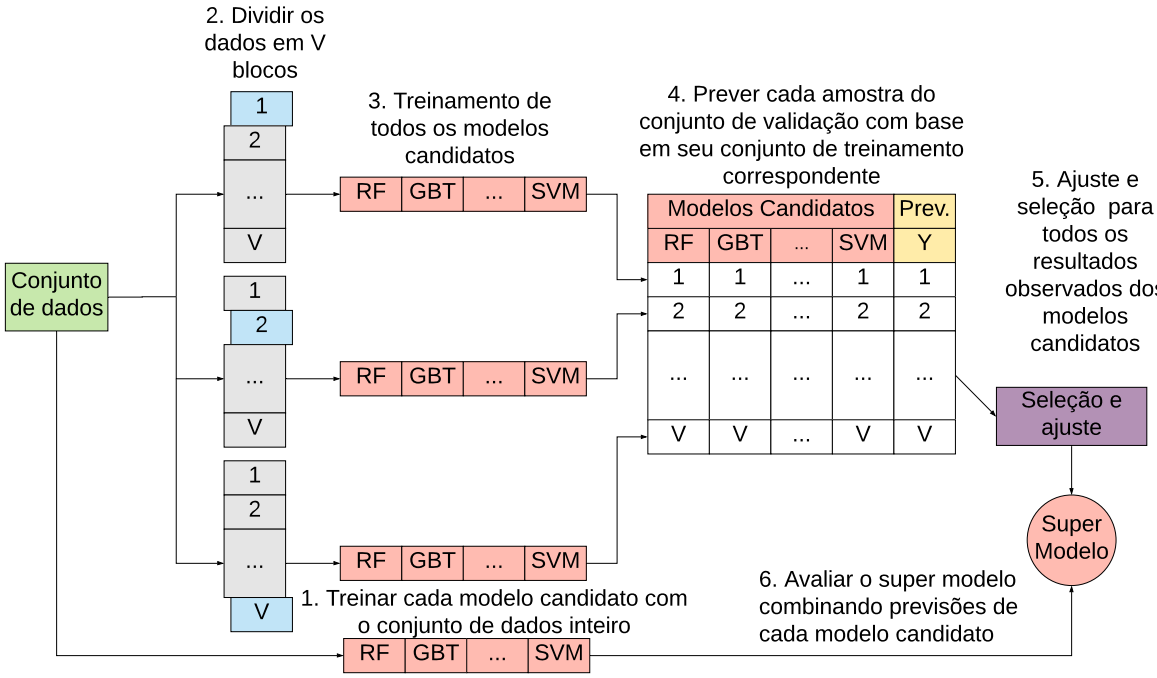


Figura 7.1: Técnica *super learner*. O conjunto de dados é dividido em blocos de V para treinar modelos candidatos. Os modelos candidatos realizam a previsão dos blocos de validação com base em seu bloco de treinamento correspondente. O super modelo recebe como entrada características da classificação de saída dos modelos candidatos validados pelo processo de seleção e ajuste. Adaptado de [94]

7.2 Aprendizado Incremental

Uma primeira abordagem para o tratamento de grandes massas de dados em fluxo é o uso de métodos de aprendizado que sejam capazes de aprender com dados infinitos em tempo finito. A ideia central é aplicar métodos de aprendizado que limitem a perda de informação ao usar modelos com dados finitos em relação a modelos com dados infinitos. Para tanto, a perda de informação é medida em função do número de amostras usadas em cada etapa de aprendizado e, então, o número de amostras usadas em cada etapa é minimizado mantendo-se o limiar de perda conservado.

A resolução do problema de o quanto de informação se perde ao diminuir o número de amostras é dada usando o limite de Hoeffding (*Hoeffding bound*) [29]. Considerando uma variável aleatória real x, cujo valor está contido no intervalo R, supõe-se que são feitas n observações independentes da variável e computa-se a média $\bar{r}$. O *Hoeffding bound* garante que, com probabilidade $1 - \delta$, a média verdadeira da variável é dada por, pelo menos, $\bar{x} - \epsilon$, em que

$$\epsilon = \sqrt{\frac{R^2 ln(1/\delta)}{2n}}. \tag{7.1}$$

O *Hoeffding bound* é independente da distribuição que gera a variável x. A partir desse resultado, desenvolvem-se os algoritmos de aprendizado de máquina para treinamento com fluxos de dados. Contudo, vale ressaltar que se assume que os valores gerados pela variável x advêm de um processo estocástico estacionário. Nos casos em que há uma mudança no processo que gera a variável usada no treinamento de métodos de aprendizado por fluxo, é dito que ocorre uma mudança de conceito (*concept drift*) e, portanto, faz-se necessário um novo treinamento do método de aprendizado [96].

As abordagens típicas para aprender novas informações envolvem a manutenção do comportamento estocástico dos dados ou o descarte do classificador existente e, consequentemente, o retreinamento com os dados acumulados até o momento. As abordagens que consideram o fim da estabilidade estatística dos dados, ou seja, o processo deixa de ser estocástico, resultam na perda de todas as informações adquiridas anteriormente, o que é conhecido como o esquecimento catastrófico. Dessa forma, Polikar *et al.* definem que algoritmos de aprendizado incremental devem satisfazer os requisitos de obter informações adicionais de novos dados; de não exigir acesso aos dados originais, usados para treinar o classificador já existente; de preservar o conhecimento previamente adquirido, ou seja, não deve sofrer esquecimento catastrófico; e de ser capaz de acomodar novas classes que possam ser introduzidas por novos dados. Assim, classificadores que adotem o aprendizado incremental não requerem o treinamento de todo o classificador no caso de uma mudança no comportamento estacionário dos dados em fluxo.

7.2.1 Algoritmos em linha de árvores de decisão incrementais

Esses algoritmos são divididos em duas categorias: i) árvores construídas através de um algoritmo guloso de busca, em que a adição de novas informações envolvem a reestruturação completa da árvore de decisão e ii) árvores incrementais que mantêm um conjunto suficiente de estatísticas em cada nó da árvore para realizar um teste de divisão do nó, tornando a classificação mais específica, quando as estatísticas acumuladas no nó são favoráveis à divisão. Um exemplo desse tipo de árvore incremental é o sistema *Very Fast Decision Tree* (VFDT). O objetivo do sistema VFDT é projetar um método de aprendizado por árvore de decisão para conjuntos de dados extremamente grandes, potencialmente infinitos. A ideia central é que cada amostra de informação seja lida uma única vez e em um pequeno tempo de processamento. Isso possibilita o gerenciamento direto das fontes de dados em linha (*online*), sem armazenar as amostras. Para encontrar a melhor característica que deve ser testado em um determinado nó, pode ser suficiente considerar apenas um pequeno subconjunto das amostras de treinamento que passam por esse nó. Assim, dado um fluxo de amostras, as primeiras serão usadas para escolher o teste da raiz; uma vez que a característica da raiz é escolhida, as amostras subsequentes são transmitidas aos nós folhas correspondentes e usadas para escolher as características apropriados nesses nós, e assim por diante, recursivamente.

Em um sistema VFDT, a árvore decisão é aprendida recursivamente, substituindo folhas por nós de decisão. Cada folha armazena estatísticas suficientes sobre valores de características. Estatísticas suficientes são aquelas necessárias por uma função de avaliação heurística que avaliam o mérito de testes de divisão de nós baseados em valores de características. Quando uma amostra está disponível, ela atravessa a árvore da raiz até uma folha, avaliando a característica apropriada em cada nó e seguindo o ramo correspondente ao valor da característica na amostra. Quando a amostra chega à folha, as estatísticas são atualizadas. Então, as condições possíveis baseadas nos valores das características são avaliadas. Caso haja suporte estatístico suficiente em favor de um teste de valor de uma característica em relação aos demais, a folha é convertida em nó de decisão. O novo nó de decisão vai ter tantos descendentes quantos valores possíveis para a característica de decisão escolhida. Os nós de decisão mantêm somente as informações sobre o teste de divisão instalado no nó. O estado inicial da árvore consiste em uma folha única que é a raiz da árvore. A função de avaliação heurística é o Ganho de Informação (*Information Gain*), denotado por $H(\cdot)$. As estatísticas suficientes para estimar o mérito de uma característica nominal são os contadores n_{ijk} que representam o número de exemplos da classe k que chegam à folha, em que a característica j recebe o valor i. O ganho de informação mede a quantidade de informação necessária para classificar uma amostra que chega ao nó: $H(A_j) = info(amostras) - info(A_j)$. A informação da característica j é dada por

$$info(A_j) = \sum_i P_i \left(\sum_k -P_{ik} log_2(P_{ik}) \right), \tag{7.2}$$

em que $P_{ik} = \frac{n_{ijk}}{\sum_a n_{ajk}}$ é a probabilidade de se observar o valor de a característica i dada a classe k e $P_i = \frac{\sum_a n_{ija}}{\sum_a \sum_b n_{ajb}}$ é a probabilidade de observar o valor da característica i.

No sistema VFDT, usa-se o limiar de Hoeffding, Equação 7.1, para decidir quantas amostras são necessárias observar antes de instalar um teste de separação em cada folha. Sendo $H(\cdot)$ a função de avaliação da característica, para o ganho de informação, $H(\cdot)$ é o $log_2(||K||)$, em que K é o conjunto de classes. Seja x_a a característica com maior valor de $H(\cdot)$, x_b a característica com o segundo maior valor de $H(\cdot)$ e $\overline{\Delta H} = \overline{H}(x_a) - \overline{H}(x_b)$, a diferença entre as duas melhores características. Então, se $\overline{\Delta H} > \epsilon$, com n amostras observadas na folha, o limiar de Hoeffding define com probabilidade $1 - \delta$ que x_a é realmente a característica com o maior valor na função de avaliação. Assim, a folha deve ser transformada em um nó de decisão que divide em x_a.

A avaliação da função para cada amostra pode ser muito custosa e, portanto, não é eficiente computar $H(\cdot)$ na chegada de cada nova amostra. A proposta VFDT só computa a função de avaliação da característica quanto um número mínimo de amostras, definido pelo usuário, são observadas desde a última avaliação. Quando duas ou mais características têm os mesmos valores de $H(\cdot)$ continuamente, mesmo com um grande

número de amostras, o limiar de Hoeffding não é capaz de decidir entre eles. Então, o VFDT introduz uma constante τ em que se $\overline{\Delta H} < \epsilon < \tau$, então a folha é convertida em um nó de decisão e o teste de decisão é baseado na melhor característica. Gama *et al.* generalizam o funcionamento do sistema VFDT para características numéricas [29].

7.2.2 *Naive Bayes* Incremental

Dado um conjunto de treinamento $\chi = (\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_n)$, em que $\mathbf{x} \in \mathbb{R}^D$ são amostras com $D - dimensionais$ no espaço de características e $y \in 1, ..., K$ são as classes correspondentes para um problema de classificação em K classes, formula-se o Teorema de Bayes como

$$p(y = i|x) = \frac{p(i)p(\mathbf{x}|i)}{p(\mathbf{x})}, \tag{7.3}$$

em que $p(i)$ é a probabilidade *a priori* de ocorrência de uma amostra da classe e $p(y|\mathbf{x})$ é a distribuição de probabilidades desconhecida do espaço de características $\mathbf{x}$ e marcada com a classe i. Uma estimativa para a distribuição desconhecida é assumir a independência das características dada a marcação da classe, levando a

$$p(x_1, x_2, ..., x_D|i)\ p(x_1|i)p(x_2|i)...p(x_D|i), \tag{7.4}$$

em que x_d representa a d-ésima dimensão no vetor de características $\mathbf{x}$. Assim, o classificador bayesiano é descrito como

$$F(\mathbf{x}) = arg \max_i \prod_{d=1}^{D} p(x_d|i). \tag{7.5}$$

Assim, a classificação é calculada fazendo a multiplicação de todas as probabilidades das classes para o valor das características da amostra [32].

A versão incremental do classificador bayesiano prevê a atualização dos valores das probabilidades das classes por características conforme novas amostras são processadas. Uma abordagem para permitir o armazenamento eficiente das funções de probabilidade conforme as amostras chegam é realizar a discretização e armazenar histogramas das características. A proposta *Incremental Flexible Frequency Discretization* (IFFD) apresenta um método para discretização de características quantitativas em uma sequência de intervalos de tamanhos flexíveis. Essa abordagem permite a inserção e a divisão de intervalos.

7.2.3 Aprendizado Incremental por Agregados de Classificadores

O algoritmo ARTMAP baseia-se na geração de novos agrupamentos de decisão em resposta a novos padrões que são suficientemente diferentes de instâncias vistas anteriormente. O valor de quanto diferente é um padrão já conhecido de um novo é controlado por um parâmetro de vigilância definido pelo usuário. Cada agrupamento aprende em um hiper-retângulo que é uma porção diferente do espaço de características, em um modo não supervisionado, que são então mapeados para classes alvo. Como os agrupamentos são sempre mantidos, o ARTMAP não sofre o esquecimento catastrófico. Além disso, ARTMAP não requer acesso a dados previamente vistos e pode acomodar novas classes. Contudo, o ARTMAP é muito sensível à seleção do parâmetro de vigilância, aos níveis de ruído nos dados de treinamento e à ordem em que os dados de treinamento chegam.

O algoritmo AdaBoost (*adaptive boosting*) gera um conjunto de hipóteses e as combina através da votação da maioria ponderada das classes previstas pelas hipóteses individuais. As hipóteses são geradas pelo treinamento de um classificador fraco*, usando instâncias extraídas de uma distribuição atualizada periodicamente dos dados de treinamento. Esta atualização de distribuição garante que instâncias mal classificadas pelo classificador anterior sejam mais provavelmente incluídas nos dados de treinamento do próximo classificador. Assim, os dados de treinamento de classificadores consecutivos são voltados para instâncias cada vez mais difíceis de classificar.

O algoritmo de aprendizagem incremental Learn ++ é inspirado pelo AdaBoost, originalmente desenvolvido para melhorar o desempenho de classificação de classificadores fracos. Em essência, Learn ++ gera um conjunto de classificadores fracos, cada um treinado usando uma distribuição diferente de amostras de treinamento. As saídas desses classificadores são, então, combinadas usando o regime de votação por maioria para obter a regra final de classificação. O uso de classificadores fracos é interessante, pois a instabilidade para que construam suas decisões é suficiente para que cada decisão seja diferente das demais, para que pequenas modificações em seus conjuntos de dados de treinamento sejam refletidas em classificações distintas.

7.3 A Técnica Proposta

Na proposta super aprendizado incremental, é feita uma mesclagem da variante do *super learner* e, para gerar o super modelo, redes neurais com aprendizado incremental. Na variante do *super learner*, é feita a divisão do conjunto de dados em um conjunto de treinamento e um conjunto de teste. Em seguida, utiliza-se o conjunto de treinamento para treinar os modelos candidatos. No caso do MineCap, foram utilizados os algoritmos

*Algoritmos de classificação que a acurácia é próxima à classificação aleatória.

Floresta Aleatória e *Gradient Boosted Tree*, que foram os modelos que tiveram melhor desempenho, como pode ser visto no Capítulo 8. Ambos foram treinando com o mesmo conjunto de dados. Cada amostra do conjunto de dados de entrada para o super modelo tem como características as probabilidades da amostra ser da classe fluxo normal, classificado como 0, ou fluxo de mineração, classificado como 1, geradas a partir das previsões feitas dos modelos candidatos aplicados sobre o conjunto de dados de teste.

Os dois modelos candidatos geram a lista $W = (w_1, w_2, w_3, w_4, y)$ em que w_1 é a probabilidade do primeiro modelo candidato retornar como saída a classe tráfego normal, w_2 é a probabilidade do primeiro modelo candidato retornar como saída a classe de trafego de mineração, w_3 e w_4 têm a mesma representação para o segundo modelo candidato. Finalmente, y é a saída desejada de cada amostra, ou seja, a classe alvo dessa amostra. A lista W gerada pelos modelos candidatos é passada como entrada para treinar a rede neural incremental utilizando o algoritmo de Perceptron Multi-Camadas MLP. A Figura 7.2 mostra o diagrama da técnica de super aprendizado incremental desenvolvida no MineCap.

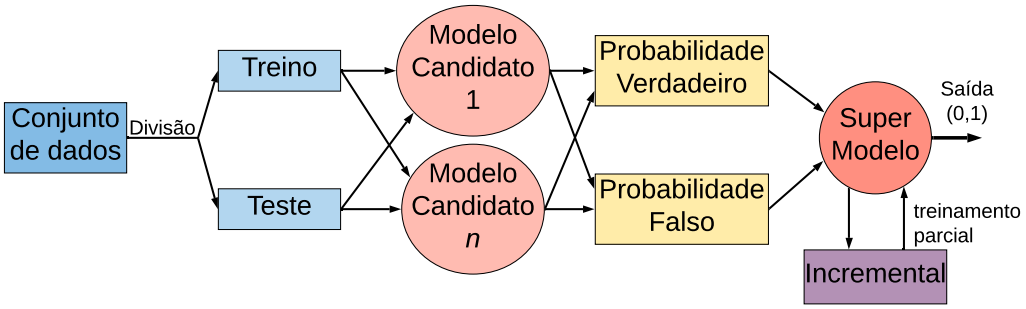


Figura 7.2: O super aprendizado incremental divide o conjunto de dados em um bloco de treinamento e bloco de teste. Os modelos candidatos são treinados com o bloco de treinamento. O super modelo recebe as probabilidades de cada amostra, em vez de somente a classe de saída. O super modelo é treinado de forma incremental com novas amostras coletadas.

Quando o MineCap recebe novos dados, os envia aos modelos candidatos para classificação e, em seguida, armazena as saídas em um arquivo chamado *incremental*. Somente depois, envia os dados como entrada para a rede neural para realizar a previsão. A rede neural classifica cada amostra recebida em linha e anexa à saída prevista à entrada da amostra no arquivo *incremental*. Cada entrada do arquivo *incremental* possui a lista $W = (w_1, w_2, w_3, w_4, y)$, em que todos os w são a saída dos modelos candidatos, floresta aleatória e *Gradient Boosted Tree* e, no caso do MineCap, y é a saída do super modelo. Quando um número k suficiente de amostras é acumulado, o processo incremental verifica cada amostra e coleta apenas as amostras que possuem maior probabilidade de serem fluxo de mineração e menor probabilidade de serem fluxo normal ou o inverso para depois realizar a aprendizagem parcial. A quantidade k de amostras acumuladas para o treinamento parcial é proporcional ao tempo para a aprendizagem com novos dados. Quanto maior a quantidade, maior o tempo para a aprendizagem.

No protótipo do mecanismo proposto, é usado o valor de $k = 50$, pois verificou-se que com 50 amostras há o compromisso de executar o treinamento parcial do super modelo entre 1 e 2 minutos, no caso de ocorrência constante de mineração na rede.

Capítulo 8

O Sistema MineCap

O sistema MineCap é executado em uma estação (*host*) ou em um aglomerado computacional (*cluster*) separado do controlador de rede. A execução do MineCap em uma estação separada do controlador de rede evita a sobrecarga no controlador. O controlador da rede redireciona todos os pacotes de saída da rede local para o MineCap, aplicando a técnica de espelhamento de porta no *gateway* de rede. Conforme mostrado na Figura 8.1, o mecanismo proposto apresenta uma arquitetura de três camadas: captura, processamento e bloqueio.

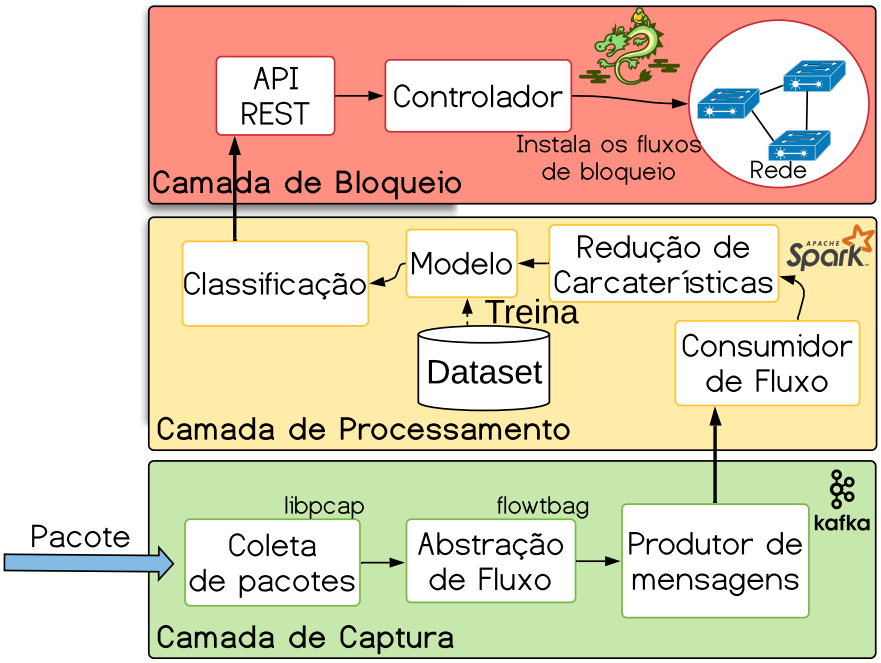


Figura 8.1: A arquitetura da ferramenta MineCap. Os pacotes chegam na interface de rede, são abstraídos em fluxos que, por sua vez, são publicados no serviço de mensagens Apache Kafka. O Spark Streaming consome a mensagem, pré-processa os dados e envia para o modelo treinado para classificação.

8.1 Camadas do MineCap

A **camada de captura** tem como objetivo capturar os dados e prepará-los para a camada superior. O primeiro passo é capturar os pacotes de rede através da execução da biblioteca `libpcap`. Esses pacotes capturados são resumidos em fluxos de rede. Define-se fluxo de rede como a sequência de pacotes que possuem o mesmo IP de origem, IP de destino, porta de origem, porta de destino e protocolo de transporte. Foi desenvolvido uma aplicação na linguagem Python, baseada na aplicação `flowtbag`*, para realizar a abstração de pacotes em fluxos. Os fluxos são publicados no serviço de mensagens Apache Kafka, um sistema de publicação e assinatura (*publisher/subscriber*) que serve como uma central de produção de dados em fluxo distribuído, garante o armazenamento e a entrega confiável das mensagens. O Kafka entrega as mensagens produzidas na camada de captura para a camada de processamento.

A **camada de processamento** é responsável por consumir, processar e classificar os fluxos da rede que o Apache Kafka fornece como dados em fluxo. O mecanismo MineCap adota o Apache Spark como sua plataforma de processamento de fluxo em linha. O Apache Spark Streaming apresenta melhor desempenho em relação à tolerância a falhas quando comparado a outras plataformas de processamento de fluxo [58], adicionando robustez e resiliência ao processamento. O MineCap evita assim a perda de informação. O Spark consome o conteúdo do Kafka com a biblioteca Spark Streaming. O Apache Kafka é um intermediário de mensagens que entrega mensagens aos processos assinantes, mas não suporta execução de algoritmos mais complexos, como o aprendizado de máquina. O MineCap incorpora o algoritmo de Análise de Componente Principal PCA [58], que reduz um grande conjunto de características a um conjunto menor de características artificiais que ainda contém a maior parte da quantidade de informação do conjunto original. A `MLlib` é a biblioteca de aprendizado de máquina escalável do Spark que consiste em algoritmos e utilitários comuns para o aprendizado de máquina. Serão avaliados quatro algoritmos de aprendizado de máquina: Floresta Aleatória, *Gradient Boosted Tree*, *Naive Bayes* e Regressão Logística. Os dois melhores algoritmos serão utilizados como modelos candidatos na técnica de super aprendizado incremental.

A **camada de bloqueio** recebe a saída do classificador e instala uma regra de bloqueio para fluxos rotulados como mineração de criptomoeda. O OpenFlow 1.3 [4] é o protocolo de rede definida por *software* utilizado pelo MineCap. O Ryu† é baseado em Python e foi escolhido como o controlador SDN para o protótipo desenvolvido, devido à sua facilidade de implantação e baixo tempo para o desenvolvimento de aplicações. No entanto, qualquer outro controlador poderia ser utilizado. A integração entre o MineCap e o controlador é agnóstica, o que possibilita a integração ou atualização de outros controladores SDN. O MineCap se comunica com o controlador por meio de um interface

*Disponível em https://github.com/DanielArndt/flowtbag.
†Disponível em https://osrg.github.io/ryu/.

de Transferência de Estado Representacional, *Representational State Transfer* (REST) executada no controlador, que recebe uma mensagem criptografada, na qual os fluxos de mineração são identificados e, então, as regras de bloqueio são instaladas na rede. Os fluxos de bloqueio instalados nos *switches* OpenFlow durante os testes efetuados são de 5 minutos, esse parâmetro é variável e pode ser alterado conforme necessário. Esse tempo foi escolhido, pois os testes gerados ocorreram em períodos de 30 minutos, um tempo de bloqueio curto gera mais tráfego de mineração na rede, assim, gerando mais eventos de bloqueio.

8.2 A Avaliação

Um protótipo do MineCap foi desenvolvido para avaliar a proposta. Os experimentos foram realizados em um computador equipado com um processador Intel Core i7 7700 a 3.60 GHz, com oito núcleos e 16 GB de RAM. A avaliação foi realizada em uma rede emulada, usando a plataforma de emulação `Mininet`, com 16 *hosts* em uma topologia em árvore personalizada, usando sete comutadores executando o protocolo OpenFlow 1.3 [4], como ilustrado na Figura 8.2.

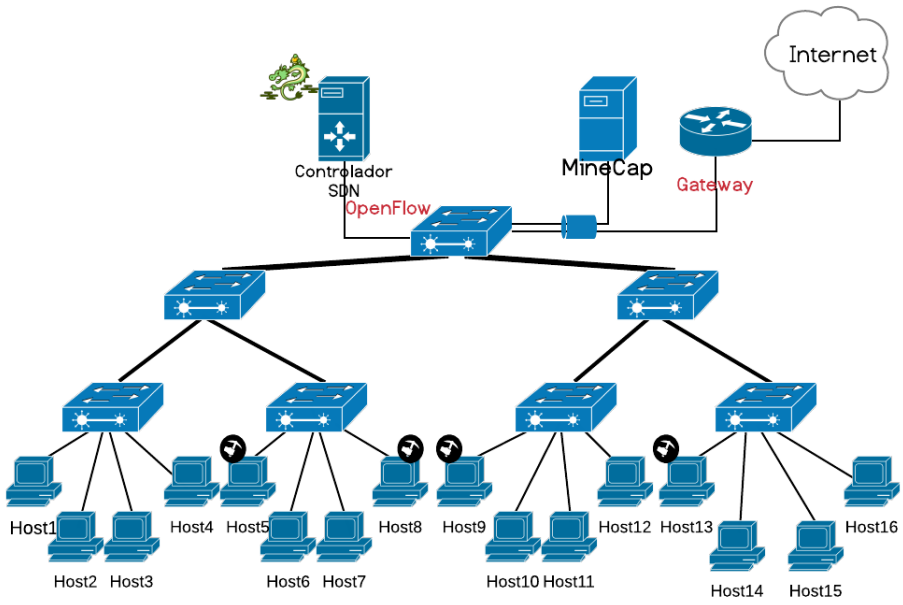


Figura 8.2: Topologia utilizada nos testes de avaliação. Utilizou-se uma rede emulada pelo Mininet, o número de mineradores varia de acordo com o teste efetuado. A topologia possui sete *switches* em árvore com dezesseis *hosts*.

Na primeira avaliação, entre os 16 *hosts* do ambiente, quatro deles executavam aplicativos de mineração de criptomoedas[‡] e o restante estava repetindo o tráfego de 30 minutos

[‡]O tráfego de mineração usado para treinar os algoritmos de aprendizado de máquina se originam da execução dos aplicativos de mineração `cpuminer` e `xmrig`.

contido em um arquivo de captura real de tráfego de rede, usando `tcpreplay`[§]. Nessa etapa, é avaliada a diferença entre os dois melhores modelos dentre os demais. O tráfego contido no arquivo de captura foi previamente classificado para ser comparado com a saída de modelos de aprendizado de máquina, para ser utilizado como linha de base. Utilizou-se *pools* de mineração com portas TCP conhecidas, para posterior classificação manual do conjunto de dados utilizado para treinar os modelos candidatos. Nos testes de avaliação, utilizaram-se *pools* diferentes das utilizadas no conjunto de dados de treinamento. O *gateway* de rede teve sua porta espelhada para o MineCap para garantir que todo o tráfego de rede passasse pela classificação. O MineCap instala um fluxo no controlador SDN bloqueando todo o tráfego classificado como mineração de criptomoeda e, portanto, os pacotes são rejeitados diretamente no comutador OpenFlow da rede. O conjunto de dados utilizado para treinar e testar os modelos foi criado no laboratório MídiaCom, em um ambiente estilo "mundo fechado", em que o tráfego foi capturado em um ambiente controlado com um comutador, computadores atuando como mineradores e outros utilizando diferentes perfis de navegação como *streaming* de vídeo, descarga de arquivos, acesso a sítios Web e outros. Os aplicativos de mineração utilizados foram o MinerGate[¶] e o GuiMiner[‖], executando em máquinas com sistema operacional Windows. Antes de particionar o conjunto de dados em conjunto de treinamento e teste, o conjunto de dados foi embaralhado para evitar qualquer elemento de viés ou padrões nos conjuntos de dados. Assim, com o conjunto de dados embaralhado, a qualidade e o desempenho dos modelos são melhorados. O conjunto de dados passou pelo processo de *oversampling*, pois a técnica é frequentemente usada para balancear a quantidade de classes do conjunto de dados [63]. O conjunto de dados criado possui abstrações de fluxo de rede, que são conjuntos de pacotes com a mesma quíntupla: IP de origem, porta de origem, IP de destino, porta de destino e protocolo de transporte. No total cada amostra possui 46 características. Para evitar que os modelos aprendam os endereços IP e portas utilizadas no conjunto de dados, essas características são removidas na etapa de pré-processamento. As *pools* de mineração utilizadas para avaliar os modelos de aprendizado de máquina são diferentes das utilizadas no conjunto de dados. A Tabela 8.1 possui a descrição de cada característica.

A Figura 8.3 mostra a curva de características operacionais do receptor ROC, a precisão, a sensibilidade e a especificidade de cada algoritmo de classificação testado. A curva ROC mede e especifica o desempenho dos algoritmos testados através do relacionamento entre verdadeiros positivos e falsos positivos em diversos pontos de corte na probabilidade de uma amostra pertencer a uma classe. É um método gráfico robusto e direto que permite estudar a variação da sensibilidade e especificidade para diferentes valores de corte.

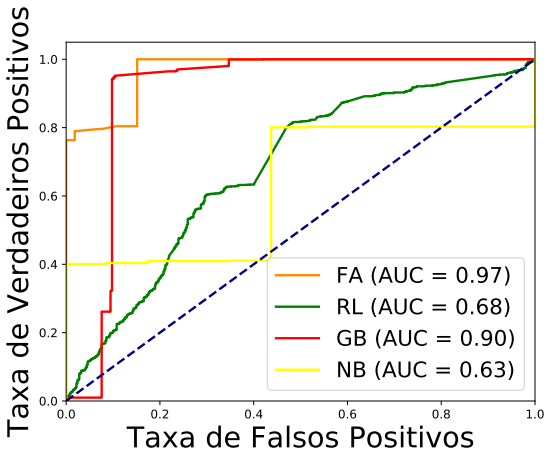
Os algoritmos de aprendizado de máquina Floresta Aleatória e *Gradient Boosted Tree* superaram os outros algoritmos, com boa precisão e sensibilidade. Regressão Logística

[§]Disponível em https://github.com/appneta/tcpreplay.
[¶]Disponível em https://minergate.com.
[‖]Disponível em https://guiminer.org.

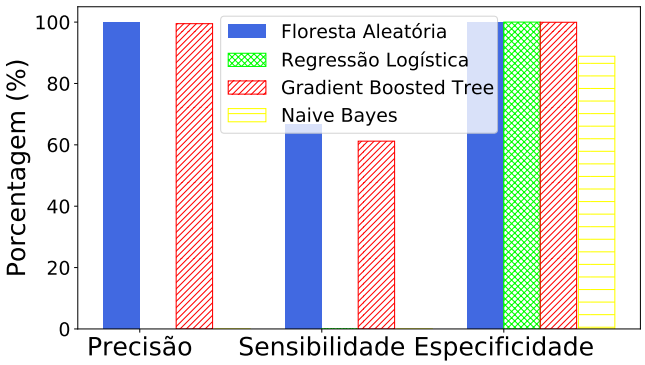
ID	Nome	Descrição
1	`srcip`	IP de Origem
2	srcport	Número da Porta de Origem
3	dstip	IP de Destino
4	dstport	Número da Porta de Destino
5	proto	Protocolo de Transporte
6	total_fpackets	Total de pacotes em direção à frente
7	total_fvolume	Total de bytes em direção à frente
8	total_bpackets	Total de pacotes em direção contrária
9	total_bvolume	Total de bytes em direção contrária
10	min_fpktl	Tamanho menor pacotes em direção à frente
11	mean_fpktl	Tamanho médio dos pacotes em direção à frente
12	max_fpktl	Tamanho maior pacotes em direção à frente
13	std_fpktl	Desvio padrão da média dos pacotes em direção à frente
14	min_bpktl	Tamanho menor pacotes em direção contrária
15	mean_bpktl	Tamanho médio dos pacotes em direção contrária
16	max_bpktl	Tamanho maior pacotes em direção contrária
17	std_bpktl	Desvio padrão da média dos pacotes em direção contrária
18	min_fiat	Quantidade mínima de tempo entre dois pacotes na direção à frente
19	mean_fiat	Quantidade média de tempo entre dois pacotes na direção à frente
20	max_fiat	Quantidade máxima de tempo entre dois pacotes na direção à frente
21	std_fiat	Desvio padrão de tempo entre dois pacotes na Direção à frente
22	min_biat	Quantidade mínima de tempo entre dois pacotes na direção contrária
23	mean_biat	Quantidade média de tempo entre dois pacotes na direção contrária
24	max_biat	Quantidade máxima de tempo entre dois pacotes na direção contrária
25	std_biat	Desvio padrão de tempo entre dois pacotes na Direção contrária
26	duration	Duração do fluxo
27	min_active	Quantidade mínima de tempo em que o fluxo estava ativo
28	mean_active	Quantidade média de tempo em que o fluxo estava ativo
29	max_active	Quantidade máxima de tempo em que o fluxo estava ativo
30	std_active	Desvio padrão de tempo em que o fluxo estava ativo
31	min_idle	Tempo mínimo para o fluxo se tornar ativo
32	mean_idle	Tempo médio para o fluxo se tornar
33	max_idle	Tempo máximo para o fluxo se tornar ativo
34	std_idle	Desvio padrão para o fluxo se tornar ativo
35	sflow_fpackets	Número médio de pacotes em um subfluxo na direção à frente
36	sflow_fbytes	Número médio de bytes em um subfluxo na direção à frente
37	sflow_bpackets	Número médio de pacotes em um subfluxo na direção contrária
38	sflow_bbytes	Número médio de bytes em um subfluxo na direção contrária
39	fpsh_cnt	Número de vezes que a flag PSH foi utilizado na direção à frente
40	bpsh_cnt	Número de vezes que a flag PSH foi utilizado na direção contrária
41	furg_cnt	Número de vezes que a flag URG foi utilizado na direção à frente
42	burg_cnt	Número de vezes que a flag URG foi utilizado na direção contrária
43	total_fhlen	Total de bytes usados para cabeçalhos na direção à frente
44	total_bhlen	Total de bytes usados para cabeçalhos na direção contrária
45	dscp	Ponto de Código de Serviços Diferenciados
46	classe	Rótulo da classe

Tabela 8.1: Tabela de descrição do conjunto de dados criado para treinar os modelos.

e *Naive Bayes* tiveram 0% de precisão e sensibilidade, por isso ficaram sem barras na Figura 8.3(b). A Regressão Logística classificou cada fluxo como fluxo normal e o *Naive Bayes* classificou alguns fluxos como normal e outros como fluxos de mineração de criptomoedas, assim, eles não serão utilizados como modelos candidatos do super



(a) Curva de características operacionais do receptor (ROC).



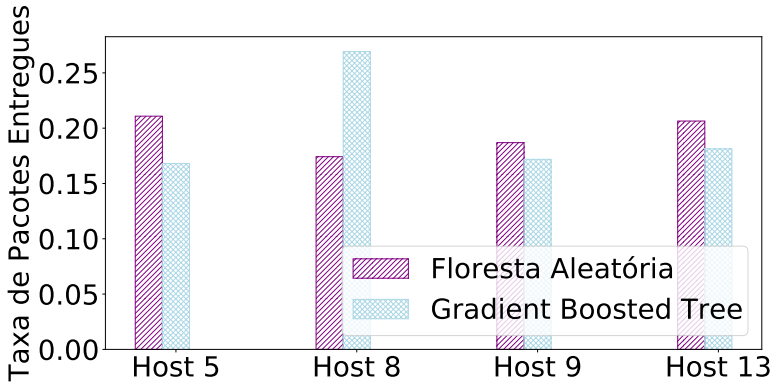
(b) Precisão, sensibilidade e especificidade dos algoritmos de classificação avaliados.

Figura 8.3: Avaliação dos algoritmos de aprendizado de máquina. a) O algoritmo de Floresta Aleatória apresenta uma Área Abaixo da Curva AUC de 0,97 e, assim, apresenta a melhor relação de compromisso entre sensibilidade e especificidade. b) *Gradient Boosted Tree* também apresenta altas taxas de sensibilidade e especificidade, porém são inferiores às da Floresta Aleatória.

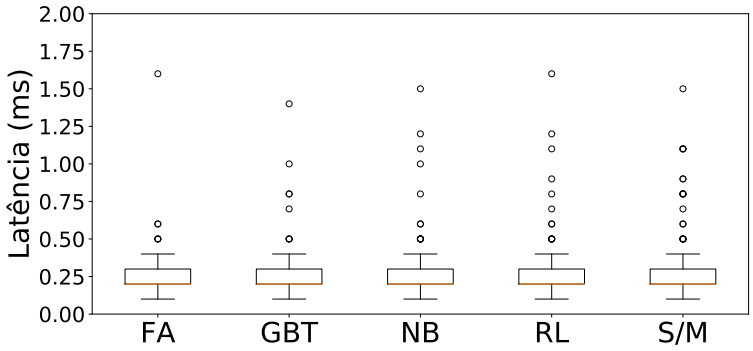
aprendizado incremental.

Além dos testes de avaliação de aprendizado de máquina, são apresentados outros dois testes de desempenho. O primeiro é a relação dos pacotes entregues e os pacotes gerados. O segundo consiste em um teste de latência da rede para verificar se o MineCap gera sobrecarga. Como os algoritmos de aprendizado de máquina *Naive Bayes* e Regressão Logística obtiveram mal desempenho, apenas os algoritmos de Floresta Aleatória e *Gradient Boosted Tree* são apresentados nas próximas avaliações. A Figura 8.4(a) mostra a taxa de tráfego entregue para os quatro *hosts* que estavam minerando criptomoedas. Vale a pena observar que o algoritmo de Floresta Aleatória bloqueia pelo menos 80% do tráfego de mineração, enquanto o *Gradient Boosted Tree* atinge 25% do tráfego de mineração entregue

no `Host 8`. Nos processos de teste, os *hosts* mineraram em *pools* distintas, assim, podendo ter desempenhos diferenciados. Em média, o algoritmo de Floresta Aleatórias bloqueia mais tráfego na rede do que o *Gradient Boosted Tree*, devido sua maior capacidade de generalização do conhecimento obtido. Destaca-se também que os algoritmos baseados em árvore tiveram melhor desempenho do que outros devido à natureza discreta dos dados de rede [58].



(a) Taxa de pacotes de mineração de criptomoedas entregues em um tráfego gerado de 30 min.



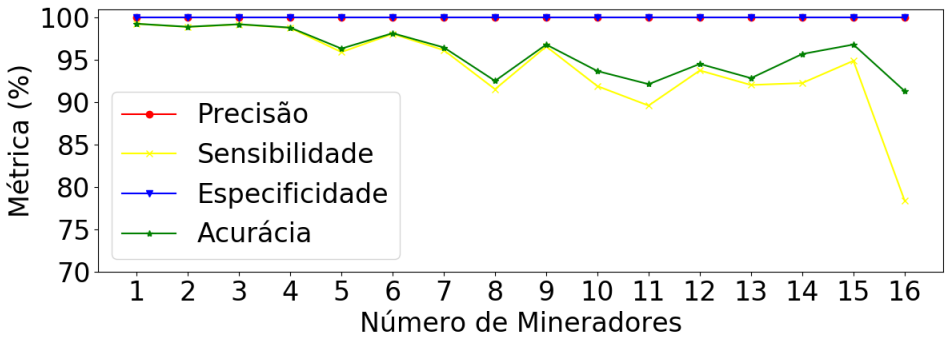
(b) Latência de comunicação no cenário avaliado.

Figura 8.4: Avaliação da latência na rede com e sem a ferramenta MineCap e gráfico da taxa de pacotes de mineração entregue. a) O classificador Floresta Aleatória bloqueou 80% do tráfego de mineração enquanto o *Gradient Boosted Tree* encaminhou mais que 25% do tráfego de mineração. b) O MineCap não acrescenta latência na rede. Floresta Aleatória (FA), *Gradiente Boosted Tree* (GBT), *Naive Bayes* (NB), Regressão Logística (RL), Sem MineCap (S/M).

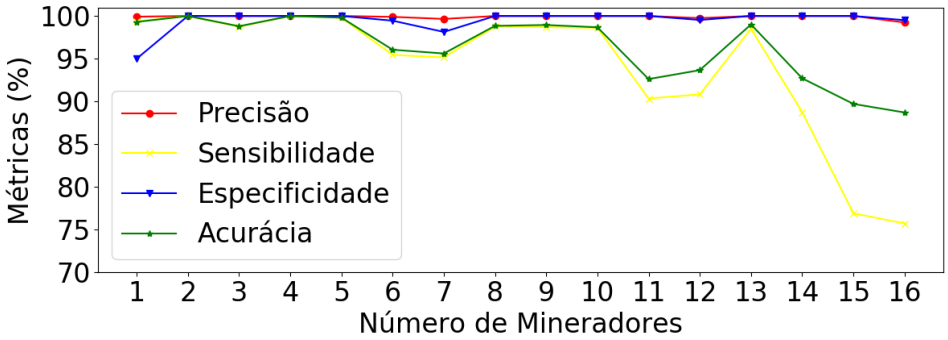
A latência da rede foi analisada enquanto os testes eram realizados gerando pacotes *Internet Control Message Protocol* (ICMP) de um *host* na rede para o controlador. O teste de latência foi executado nos testes com todos os algoritmos e, também, em um novo cenário reproduzindo o mesmo tráfego sem intervenção do MineCap. Os resultados mostram que o mecanismo MineCap não implica mais latência no encaminhamento de pacotes e, assim, mantém o desempenho da rede do cenário sem a adoção do MineCap,

como mostra a Figura 8.4(b). Também foi avaliado, separadamente, o tempo de resposta de chamada REST que foi insignificante, mostrando um atraso de menos de 1 ms em uma rede local, pois o tamanho dos pacotes nas chamadas é muito pequeno. O tempo de bloqueio é em média de 2 minutos, usando ambos os algoritmos, Floresta Aleatória e *Gradient Boosted Tree*, porque às vezes o algoritmo classifica fluxos de mineração de criptomoedas como fluxos normais. Os modelos de aprendizado de máquina são passíveis de falhas, já que falsos positivos e negativos existem nos problemas de aprendizado de máquina em que não há sobreajuste [76].

Floresta Aleatória e *Gradient Boosted Tree* obtiveram resultados similares, porém os testes anteriores utilizavam uma quantidade estática de mineradores. É importante avaliar se o aumento na quantidade de mineradores na rede impacta o desempenho dos classificadores. Assim, o teste com 30 minutos de tráfego foi realizado variando a quantidade de mineradores.



(a) Precisão, sensibilidade, especificidade e precisão do modelo Floresta Aleatória para cenários com diferentes números de mineradores.



(b) As mesmas métricas foram utilizadas para o modelo *Gradient Boosted Tree* no cenários com diferentes números de mineradores.

Figura 8.5: Avaliação dos algoritmos de aprendizado de máquina de acordo com o crescimento do número de mineradores.

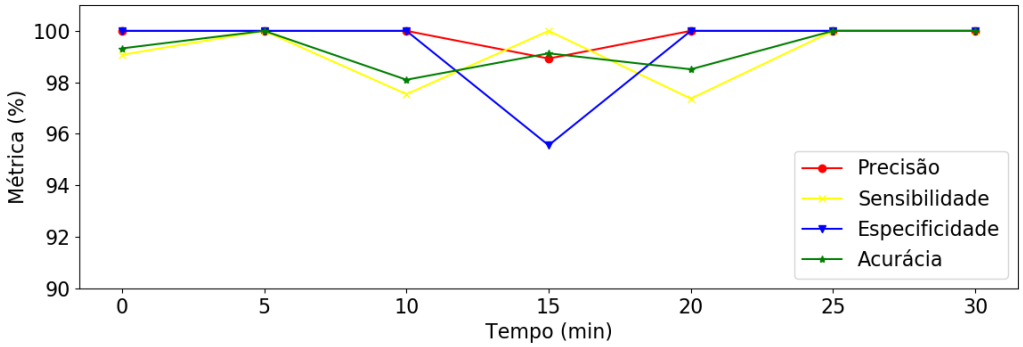
A Figura 8.5 mostra que, à medida que mais mineradores são adicionados à rede, a precisão e a sensibilidade dos modelos diminuem, mas continuam com resultados aceitáveis,

superiores a 75%. Paralelamente, é proposto uma variante do *super learner* [94] usando esses dois modelos pelos seguintes motivos: i) é desejado ter um desempenho igual ou superior ao dos algoritmos usados utilizando no *super learner*; ii) é importante o uso de algoritmos de aprendizado de máquina presentes na biblioteca MLlib [72] do Spark, pois obtêm-se melhores aproveitamentos da abstração de dados do Spark e a biblioteca não possui suporte para o aprendizado incremental. Assim, utiliza-se uma rede neural como o super modelo, pois suporta o aprendizado incremental. Para testar a eficácia do super aprendizado incremental, utilizou-se o super modelo em dois cenários diferentes. O cenário *a)* com cinco mineradores, e o cenário *b)* com quinze mineradores. Esses cenários representam ambientes com poucos mineradores e muitos mineradores, proporcionalmente ao total de dezesseis *hosts*. Em cada cenário, executou-se o mesmo padrão de 30 minutos de tráfego de rede reproduzido. Esse tempo foi dividido em fatias de cinco minutos. A cada cinco minutos são calculadas a precisão, a sensibilidade, a especificidade e a precisão do modelo. Com isso, podemos analisar o quanto o super modelo melhora ou piora conforme o tempo.

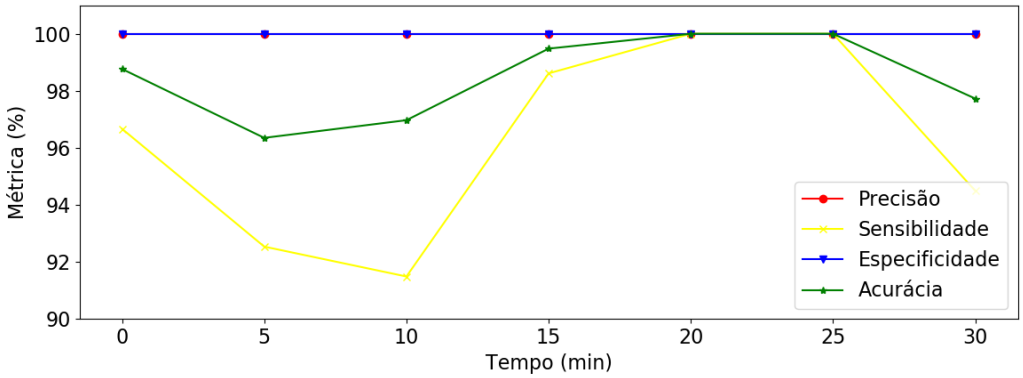
É possível verificar na Figura 8.6 que em algumas fatias de tempo o modelo se comporta mal, mas melhora quando aprende com os novos dados. Ainda, em alguns momentos, acontece o desvio de conceito, mas o super modelo se adapta, pois é treinado de forma incremental com as melhores amostras selecionadas de acordo com sua probabilidade de classificação e, então, é garantido que o super modelo aprenderá com dados que possuem alta probabilidade de estarem corretos.

Durante a avaliação da técnica de super aprendizado incremental no cenário de muitos mineradores, o processo responsável pelo super aprendizado incremental passou por uma análise de consumo de CPU e memória RAM. O consumo de CPU manteve-se baixo como pode ser visto na Figura 8.7, mantendo uma média de 6% de consumo. Porém, no processo de treinamento dos modelos candidatos, houve um pico de 70% de consumo, mas foi por um período curto, por esse motivo não apareceu no gráfico.

O super aprendizado incremental consumiu 98,3 megabytes de memória RAM, um consumo que pode ser considerado baixo nos dias atuais. Efetuou-se essa mesma coleta durante as avaliações do classificadores Árvore Aleatória e *Gradient Boosted Tree* e ambos consumiram 32,76 megabytes de memória RAM. Então, conclui-se que, para este conjunto de dados, cada classificador utilizado consome 32,76 megabytes de RAM, para o super aprendizado incremental utilizou-se dois modelos candidatos e um super modelo, totalizando 98,3 megabytes, i.e., $3x$ onde $x = 32,76$.



(a) Precisão, sensibilidade, especificidade e precisão do super aprendizado incremental com poucos mineradores durante o tempo.



(b) As mesmas métricas de avaliação do teste anterior utilizando o super aprendizado incremental com muitos mineradores durante o tempo.

Figura 8.6: Avaliação da técnica de super aprendizado incremental durante reproduzindo um tráfego de 30 min.

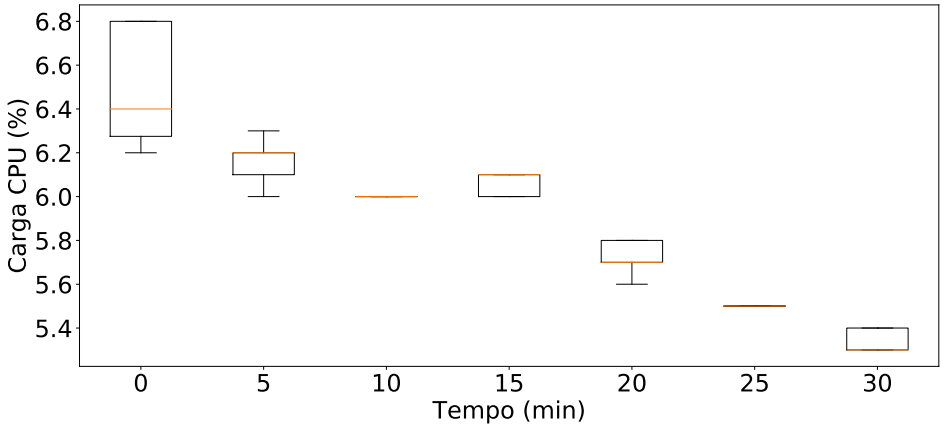


Figura 8.7: Gráfico de consumo de CPU do processo responsável pelo super aprendizado incremental.

Capítulo 9

Conclusão

A tendência da mineração de criptomoeda é crescer proporcionalmente ao valor monetário das moedas digitais. Atualmente, a mineração da criptomoeda mais popular, Bitcoin, ultrapassa de 80.000.000 TH/s*, assim, ambientes corporativos e críticos devem se proteger contra a mineração não autorizada de criptomoedas, pois o processo de mineração consome em excesso recursos importantes como de processamento, de energia e de rede. O sistema MineCap foi desenvolvido para identificar e bloquear os fluxos em linha de mineração de criptomoedas em uma rede definida por *software*. Foi desenvolvido um protótipo do sistema para avaliação. Para aumentar a velocidade de análise e melhorar a eficiência da análise de grandes massas de dados, é obrigatório implementar métodos de pré-processamento. Nesta dissertação é apresentado diferentes métodos para redução de dimensionalidade e seleção de características. Os dados em sua forma bruta apresentam baixa qualidade e ao serem diretamente processados resultam na utilização de um grande espaço de armazenamento e na baixa qualidade da informação gerada. Na etapa de pré-processamento de dados do MineCap, as características IP de origem, IP de destino, porta de origem, porta de destino e protocolo de transporte foram descartadas, a fim de não viciar o modelo com as características específicas das *pools* de mineração utilizadas no conjunto de dados. Para redução de características o MineCap incorpora o algoritmo PCA, que reduz as características do conjunto de dados para um número de k características. Foi criado, para o MineCap, um conjunto de dados criado no laboratório MídiaCom com amostras de tráfego de mineração e trafego normal com diferentes perfis de navegação. O tráfego de mineração foi feito com um *software* de mineração diferente do que foi realizado nas avaliações, utilizou-se o MinerGate e o GuiMiner, executado no sistema operacional Windows, enquanto que para a avaliação utilizou-se o xmrig e o cpuminer, executado no sistema operacional Linux. Para o tráfego normal foi coletado diversos perfis de tráfego como *download* de arquivos, *streaming* de vídeo de diversas fontes, navegação web com http e https.

Essa dissertação também propôs a técnica de super aprendizado incremental onde

*Informação disponível em https://www.blockchain.com/pt/charts/hash-rate.

modelos de aprendizado de máquina, chamados de modelos candidatos, são treinados e validados, a saída desses modelos são as probabilidades de pertinência de cada amostra a uma classe, essa probabilidade é passada para treinar o super modelo que está equipado com um algoritmo de aprendizado incremental. Após o processo de treinamento, todos os fluxos recebidos são entregues aos modelos candidatos para previsão e as probabilidades dos dois modelos são recebidas pelo super modelo. O super modelo, no caso do super aprendizado incremental, é treinado de forma incremental com novas amostras, essas amostras são novos dados coletados em linha. Somente as amostras com maiores probabilidades de serem de uma determinada classe serão escolhidas para o treinamento parcial do super modelo, assim somente as melhores amostras são expostas ao super modelo, evitando ruídos no treinamento parcial. Por fim, a avaliação realizada da técnica de super aprendizado incremental consiste em dois testes de 30 minutos, a cada 5 minutos é avaliado a acurácia, a sensibilidade, a especificidade e a precisão do super modelo. O primeiro teste ocorreu em um cenário com uma baixa quantidade de mineradores com base no total de *hosts* e o segundo teste com muitos mineradores.

Os resultados das avaliações evidenciam que a proposta não sobrecarrega a rede e o controlador em relação à latência entre os *hosts* da rede. Durante a avaliação dos modelos de aprendizado de máquina constatou-se que os algoritmos de aprendizado de máquina Floresta Aleatória e *Gradient Boosted Tree* obtiveram melhores resultados em relação aos outros algoritmos, então, incorporaram à técnica de super aprendizado incremental. Ambos apresentaram boa capacidade de generalização, com precisão e especificidade de cerca de 100% e sensibilidade de 66,5%. A técnica de super aprendizado incremental obteve bons resultados e demonstrou um correto funcionamento em relação a fatias de tempo com diferentes quantidades de mineradores na rede. A técnica aprende com novos dados, mantendo alto desempenho desde o início da execução e, em alguns casos, melhorando com o tempo.

9.1 Trabalhos Futuros

Para trabalhos futuros pretende-se aprimorar a técnica de super aprendizado incremental, viso que a técnica foi avaliada com apenas dois modelos candidatos, não é possível saber, sem mais avaliações, como o super modelo reagiria com um número maior de modelos candidatos e qual número ideal de modelos candidatos. Foi utilizado o algoritmo redes neurais como o super modelo, outros algoritmos podem ser testados e consequentemente ter resultados melhores ou piores. Não foi utilizado engenharia de características entre a saída dos modelos candidatos e a entrada do supermodelo, assim como é feito na técnica original do *Super Learner* [94].

Utilizar o código desenvolvido para o MineCap utilizando outro conjunto de dados, como por exemplo de anomalias e ataques de rede, assim pode-se avaliar se o MineCap irá

se destacar dentre outros trabalhos com o foco de detecção de intrusão.

Referências

[1] *OpenFlow Switch Specification, Version (Wire Protocol 0x01).* The OpenFlow Consortium, 2009.

[2] *OpenFlow Switch Specification Version 1.1.0 (Wire Protocol 0x02).* The OpenFlow Consortium, 2011.

[3] *OpenFlow Switch Specification Version 1.2 (Wire Protocol 0x03).* The OpenFlow Consortium, 2011.

[4] *OpenFlow Switch Specification Version 1.3.0 (Wire Protocol 0x04).* The OpenFlow Consortium, 2012.

[5] Abadi, D. J., Ahmad, Y., Balazinska, M., Cetintemel, U., Cherniack, M., Hwang, J.-H., Lindner, W., Maskey, A., Rasin, A., Ryvkina, E., others. The design of the borealis stream processing engine. In *Cidr* (2005), vol. 5, p. 277–289.

[6] Amin, R., Reisslein, M., Shah, N. Hybrid sdn networks: A survey of existing approaches. *IEEE Communications Surveys Tutorials 20*, 4 (2018), 3259–3306.

[7] Andreoni Lopez, M., Sanz, I., Menezes, D., Duarte, O., Pujolle, G. Catraca: uma ferramenta para classificação e análise tráfego escalável baseada em processamento por fluxo. *Salão de Ferramentas do XVII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais-SBSeg* (2017), 788–795.

[8] Andreoni Lopez, M., Sanz, I. J., Lobato, A. G. P., Mattos, D. M. F., Duarte, O. C. M. B. Aprendizado de máquina em plataformas de processamento distribuído de fluxo: Análise e detecção de ameaças em tempo real. In *Minicursos do SBRC2018*, 1 ed. SBC, 2018, cap. 3, p. 1–56.

[9] Arasu, A., Babcock, B., Babu, S., Datar, M., Ito, K., Motwani, R., Nishizawa, I., Srivastava, U., Thomas, D., Varma, R., others. Stream: The stanford stream data manager. *IEEE Data Eng. Bull. 26*, 1 (2003), 19–26.

[10] Armbrust, M., Xin, R. S., Lian, C., Huai, Y., Liu, D., Bradley, J. K., Meng, X., Kaftan, T., Franklin, M. J., Ghodsi, A., others. Spark sql: Relational data processing in spark. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data* (2015), ACM, p. 1383–1394.

[11] Balazinska, M., Balakrishnan, H., Stonebraker, M. Load management and high availability in the medusa distributed stream processing system. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data* (2004), ACM, p. 929–930.

[12] Bannour, F., Souihi, S., Mellouk, A. Distributed SDN control: Survey, taxonomy, and challenges. *IEEE Communications Surveys Tutorials 20*, 1 (2018), 333–354.

[13] Bostani, H., Sheikhan, M. Hybrid of anomaly-based and specification-based ids for internet of things using unsupervised opf based on mapreduce approach. *Computer Communications 98* (2017), 52 – 71.

[14] Boutaba, R., Salahuddin, M. A., Limam, N., Ayoubi, S., Shahriar, N., Estrada-Solano, F., Caicedo, O. M. A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications 9*, 1 (Jun 2018), 16.

[15] Carbone, P., Fóra, G., Ewen, S., Haridi, S., Tzoumas, K. Lightweight asynchronous snapshots for distributed dataflows. *arXiv preprint arXiv:1506.08603* (2015).

[16] Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi†, S., Tzoumas, K. Apache flink^{TM}: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 4*, 34 (2015), 28–38.

[17] Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Seidman, G., Stonebraker, M., Tatbul, N., Zdonik, S. Monitoring streams: A new class of data management applications. In *Proceedings of the 28th International Conference on Very Large Data Bases* (2002), VLDB '02, VLDB Endowment, p. 215–226.

[18] Chen, J., DeWitt, D. J., Tian, F., Wang, Y. Niagaracq: A scalable continuous query system for internet databases. In *ACM SIGMOD Record* (2000), vol. 29, ACM, p. 379–390.

[19] Chintapalli, S., Dagit, D., Evans, B., Farivar, R., Graves, T., Holderbaugh, M., Liu, Z., Nusbaum, K., Patil, K., Peng, B. J., others. Benchmarking streaming computation engines: Storm, flink and spark streaming. In *2016 IEEE international parallel and distributed processing symposium workshops (IPDPSW)* (2016), IEEE, p. 1789–1792.

[20] Cunha Neto, H. N., Fernandes, N. C., Mattos, D. M. F. Minecap: Online detection and blocking of cryptocurrency mining on software-defined networking. In *1st Blockchain, Robotics and AI for Networking Security Conference* (2019), DNAC.

[21] Dao, H., Mazel, J., Fukuda, K. Understanding abusive web resources: Characteristics and counter-measures of malicious web resources and cryptocurrency mining. In *Proceedings of the Asian Internet Engineering Conference* (New York, NY, USA, 2018), AINTEC '18, ACM, p. 54–61.

[22] De Assis, M. V. O., Novaes, M. P., Zerbini, C. B., Carvalho, L. F., Abrãao, T., Proença, M. L. Fast defense system against attacks in software defined networks. *IEEE Access 6* (2018), 69620–69639.

[23] de Oliveira, M. T., Carrara, G. R., Fernandes, N. C., Albuquerque, C. V. N., Carrano, R. C., de Medeiros, D. S. V., Mattos, D. M. F. Towards a performance evaluation of private blockchain frameworks using a realistic workload. In *2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)* (Paris, fevereiro de 2019).

[24] Dean, J., Ghemawat, S. Mapreduce: simplified data processing on large clusters. *Communications of the ACM 51*, 1 (2008), 107–113.

[25] Dean, J., Ghemawat, S. Mapreduce: simplified data processing on large clusters. *Communications of the ACM 51*, 1 (2008), 107–113.

[26] Eskandari, S., Leoutsarakos, A., Mursch, T., Clark, J. A first look at browser-based cryptojacking. In *2018 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)* (April 2018), p. 58–66.

[27] Fei-Fei, L., Fergus, R., Perona, P. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer Vision and Image Understanding 106*, 1 (2007), 59 – 70. Special issue on Generative Model Based Vision.

[28] Fernandes, N. C., Magalhaes, L. C. S. *Network Innovation Through Openflow and Sdn: Principles and Design. Chapter 5:Control and Management Software for SDNs: Conceptual Models and Practical View (ISBN 9781466572096)*, 1 ed. Fei Hu. (Org.), 2013.

[29] Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., Bouchachia, A. A survey on concept drift adaptation. *ACM computing surveys (CSUR) 46*, 4 (2014), 44.

[30] García, S., Ramírez-Gallego, S., Luengo, J., Benítez, J. M., Herrera, F. Big data preprocessing: methods and prospects. *Big Data Analytics 1*, 1 (Nov 2016), 9.

[31] Garofalakis, M., Gehrke, J., Rastogi, R. *Data Stream Management - Processing High-Speed Data Streams*. 2016.

[32] Godec, M., Leistner, C., Saffari, A., Bischof, H. On-line random naive bayes for tracking. In *2010 20th Int. Conference on Pattern Recognition* (Aug 2010), p. 3545–3548.

[33] Gonzalez, J. E., Xin, R. S., Dave, A., Crankshaw, D., Franklin, M. J., Stoica, I. GraphX: Graph processing in a distributed dataflow framework. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)* (2014), p. 599–613.

[34] Gonçalves, D., Bota, J., Correia, M. Big data analytics for detecting host misbehavior in large logs. In *2015 IEEE Trustcom/BigDataSE/ISPA* (Aug 2015), vol. 1, p. 238–245.

[35] Habeeb, R. A. A., Nasaruddin, F., Gani, A., Hashem, I. A. T., Ahmed, E., Imran, M. Real-time big data processing for anomaly detection: A survey. *International Journal of Information Management 45* (2019), 289 – 307.

[36] Ham, Y. J., Lee, H.-W. Big data preprocessing mechanism for analytics of mobile web log. *International Journal of Advances in Soft Computing & Its Applications 6*, 1 (2014).

[37] Han, J., Pei, J., Kamber, M. *Data mining: concepts and techniques*. Elsevier, 2011.

[38] Hartigan, J. A., Wong, M. A. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics) 28*, 1 (1979), 100–108.

[39] Haykin, S. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1998.

[40] Hong, G., Yang, Z., Yang, S., Zhang, L., Nan, Y., Zhang, Z., Yang, M., Zhang, Y., Qian, Z., Duan, H. How you get shot in the back: A systematical study about cryptojacking in the real world. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2018), CCS '18, ACM, p. 1701–1713.

[41] Hu, F., Hao, Q., Bao, K. A survey on software-defined network and openflow: From concept to implementation. *IEEE Communications Surveys Tutorials 16*, 4 (2014), 2181–2206.

[42] Hu, H., Wen, Y., Chua, T., Li, X. Toward scalable systems for big data analytics: A technology tutorial. *IEEE Access 2* (2014), 652–687.

[43] Idika, N., Mathur, A. P. A survey of malware detection techniques. *Purdue University 48* (2007).

[44] Ingols, K. Modeling modern network attacks and countermeasures using attack graphs. *Computer Security Applications Conference* (2009).

[45] Iqbal, M. H., Soomro, T. R. Big data analysis: Apache storm perspective. *Int. journal of computer trends and technology 19*, 1 (2015), 9–14.

[46] Isard, M., Budiu, M., Yu, Y., Birrell, A., Fetterly, D. Dryad: distributed data-parallel programs from sequential building blocks. *ACM SIGOPS operating systems review 41*, 3 (2007), 59–72.

[47] Isard, M., Budiu, M., Yu, Y., Birrell, A., Fetterly, D. Dryad: distributed data-parallel programs from sequential building blocks. *ACM SIGOPS operating systems review 41*, 3 (2007), 59–72.

[48] Jiang, W., Ravi, V. T., Agrawal, G. A map-reduce system with an alternate api for multi-core environments. In *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing* (May 2010), p. 84–93.

[49] Kala Karun, A., Chitharanjan, K. A review on hadoop — hdfs infrastructure extensions. In *2013 IEEE Conference on Information Communication Technologies* (April 2013), p. 132–137.

[50] Konoth, R. K., Vineti, E., Moonsamy, V., Lindorfer, M., Kruegel, C., Bos, H., Vigna, G. Minesweeper: An in-depth look into drive-by cryptocurrency mining and its defense. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (2018), ACM, p. 1714–1730.

[51] Lara, A., Kolasani, A., Ramamurthy, B. Network innovation using openflow: A survey. *IEEE Communications Surveys Tutorials 16*, 1 (2014), 493–512.

[52] Lee, G. M., Liu, H., Yoon, Y., Zhang, Y. Improving sketch reconstruction accuracy using linear least squares method. In *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement* (Berkeley, CA, USA, 2005), IMC '05, USENIX Association, p. 24–24.

[53] Liebchen, G., Twala, B., Shepperd, M., Cartwright, M., Stephens, M. Filtering, robust filtering, polishing: Techniques for addressing quality in software data. In *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)* (2007), IEEE, p. 99–106.

[54] Lin, I.-C., Liao, T.-C. A survey of blockchain security issues and challenges. *IJ Network Security 19*, 5 (2017), 653–659.

[55] 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 78* (2015), 13 – 21.

[56] Liu, J., Zhao, Z., Cui, X., Wang, Z., Liu, Q. A novel approach for detecting browser-based silent miner. In *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)* (June 2018), p. 490–497.

[57] Lobato, A. G. P., Lopez, M. A., Sanz, I. J., Cardenas, A. A., Duarte, O. C. M. B., Pujolle, G. An adaptive real-time architecture for zero-day threat detection. In *2018 IEEE International Conference on Communications (ICC)* (May 2018), p. 1–6.

[58] Lopez, M. A., Lobato, A. G. P., Duarte, O. C. M. B. A performance comparison of open-source stream processing platforms. In *2016 IEEE Global Communications Conference (GLOBECOM)* (Dec 2016), p. 1–6.

[59] Lopez, M. A., Mattos, D. M. F., Duarte, O. C. M. B., Pujolle, G. A fast unsupervised preprocessing method for network monitoring. *Annals of Telecommunications* (Aug 2018).

[60] Low, Y., Bickson, D., Gonzalez, J., Guestrin, C., Kyrola, A., Hellerstein, J. M. Distributed graphlab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment 5*, 8 (2012), 716–727.

[61] Low, Y., Bickson, D., Gonzalez, J., Guestrin, C., Kyrola, A., Hellerstein, J. M. Distributed graphlab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment 5*, 8 (2012), 716–727.

[62] Low, Y., Gonzalez, J. E., Kyrola, A., Bickson, D., Guestrin, C. E., Hellerstein, J. Graphlab: A new framework for parallel machine learning. *arXiv preprint arXiv:1408.2041* (2014).

[63] Luengo, J., Fernández, A., García, S., Herrera, F. Addressing data complexity for imbalanced data sets: analysis of smote-based oversampling and evolutionary undersampling. *Soft Computing 15*, 10 (2011), 1909–1936.

[64] Maglaras, L. A., Jiang, J. Intrusion detection in scada systems using machine learning techniques. In *2014 Science and Information Conference* (Aug 2014), p. 626–631.

[65] Malewicz, G., Austern, M. H., Bik, A. J., Dehnert, J. C., Horn, I., Leiser, N., Czajkowski, G. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data* (2010), ACM, p. 135–146.

[66] Mattos, D. M. F., Duarte, O. C. M. B., Pujolle, G. A resilient distributed controller for software defined networking. In *2016 IEEE International Conference on Communications (ICC)* (maio de 2016), p. 1–6.

[67] Mattos, D. M. F., Duarte, O. C. M. B., Pujolle, G. Reverse update: A consistent policy update scheme for software-defined networking. *IEEE Communications Letters 20*, 5 (maio de 2016), 886–889.

[68] McAfee, A., Brynjolfsson, E., Davenport, T. H., Patil, D., Barton, D. Big data: the management revolution. *Harvard business review 90*, 10 (2012), 60–68.

[69] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev. 38*, 2 (2008), 69–74.

[70] McNeil, P., Shetty, S., Guntu, D., Barve, G. Scredent: Scalable real-time anomalies detection and notification of targeted malware in mobile devices. *Procedia Computer Science 83* (2016), 1219 – 1225.

[71] Medeiros, D. S. V., Cunha Neto, H. N., Andreoni Lopez, M., Magalhães, L. C. S., Silva, E. F., Vieira, A. B., Fernandes, N. C., Mattos, D. M. F. Análise de dados em redes sem fio de grande porte: Processamento em fluxo em tempo real, tendências e desafios. *Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC 2019* (2019), 142–195.

[72] Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., others. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research 17*, 1 (2016), 1235–1241.

[73] Moon, T. K. The expectation-maximization algorithm. *IEEE Signal Processing Magazine 13*, 6 (Nov 1996), 47–60.

[74] Nogueira, A., Salvador, P., Blessa, F. A botnet detection system based on neural networks. In *2010 Fifth International Conference on Digital Telecommunications* (June 2010), p. 57–62.

[75] Owezarski, P. A near real-time algorithm for autonomous identification and characterization of honeypot attacks. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security* (New York, NY, USA, 2015), ASIA CCS '15, ACM, p. 531–542.

[76] Pietraszek, T., Tanner, A. Data mining and machine learning—towards reducing false positives in intrusion detection. *Information security technical report 10*, 3 (2005), 169–183.

[77] Porras, P. A., Valdes, A. Network surveillance, novembro de 20 2001. US Patent 6,321,338.

[78] Rathore, M. M., Paul, A., Ahmad, A., Rho, S., Imran, M., Guizani, M. Hadoop based real-time intrusion detection for high-speed networks. In *2016 IEEE Global Communications Conference (GLOBECOM)* (Dec 2016), p. 1–6.

[79] Rodriguez, J. D. P., Posegga, J. Rapid: Resource and api-based detection against in-browser miners. In *Proceedings of the 34th Annual Computer Security Applications Conference* (New York, NY, USA, 2018), ACSAC '18, ACM, p. 313–326.

[80] Rojas, E. From software-defined to human-defined networking: Challenges and opportunities. *IEEE Network 32*, 1 (janeiro de 2018), 179–185.

[81] Rüth, J., Zimmermann, T., Wolsing, K., Hohlfeld, O. Digging into browser-based crypto mining. In *Proceedings of the Internet Measurement Conference 2018* (New York, NY, USA, 2018), IMC '18, ACM, p. 70–76.

[82] Saied, A., Overill, R. E., Radzik, T. Detection of known and unknown ddos attacks using artificial neural networks. *Neurocomputing 172* (2016), 385 – 393.

[83] Shon, T., Kim, Y., Lee, C., Moon, J. A machine learning framework for network anomaly detection using SVM and GA. In *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop* (junho de 2005), p. 176–183.

[84] Singh, K., Guntuku, S. C., Thakur, A., Hota, C. Big data analytics framework for peer-to-peer botnet detection using random forests. *Information Sciences 278* (2014), 488 – 497.

[85] Stevanovic, M., Pedersen, J. M. An efficient flow-based botnet detection using supervised machine learning. In *2014 International Conference on Computing, Networking and Communications (ICNC)* (Feb 2014), p. 797–801.

[86] Stonebraker, M., Çetintemel, U., Zdonik, S. The 8 requirements of real-time stream processing. *SIGMOD Rec. 34*, 4 (dezembro de 2005), 42–47.

[87] Stonebraker, M., Kemnitz, G. The postgres next generation database management system. *Communications of the ACM 34*, 10 (1991), 78–92.

[88] Tahir, R., Huzaifa, M., Das, A., Ahmad, M., Gunter, C., Zaffar, F., Caesar, M., Borisov, N. Mining on someone else's dime: Mitigating covert mining operations in clouds and enterprises. In *International Symposium on Research in Attacks, Intrusions, and Defenses* (2017), Springer, p. 287–310.

[89] Tang, T. A., Mhamdi, L., McLernon, D., Zaidi, S. A. R., Ghogho, M. Deep learning approach for network intrusion detection in software defined networking. In *2016 International Conference on Wireless Networks and Mobile Communications (WINCOM)* (Oct 2016), p. 258–263.

[90] Tannahill, B. K., Jamshidi, M. System of systems and big data analytics – bridging the gap. *Computers & Electrical Engineering 40*, 1 (2014), 2 – 15. 40th-year commemorative issue.

[91] Tesauro, G. Reinforcement learning in autonomic computing: A manifesto and case studies. *IEEE Internet Computing 11*, 1 (janeiro de 2007), 22–30.

[92] Tsai, C.-F., Hsu, Y.-F., Lin, C.-Y., Lin, W.-Y. Intrusion detection by machine learning: A review. *Expert Systems with Applications 36*, 10 (2009), 11994 – 12000.

[93] Tsai, C.-W., Lai, C.-F., Chao, H.-C., Vasilakos, A. V. Big data analytics: a survey. *Journal of Big Data 2*, 1 (Oct 2015), 21.

[94] Van der Laan, M. J., Polley, E. C., Hubbard, A. E. Super learner. *Statistical applications in genetics and molecular biology 6*, 1 (2007).

[95] Wai, F. K., Lilei, Z., Wai, W. K., Le, S., Thing, V. L. L. Automated botnet traffic detection via machine learning. In *TENCON 2018 - 2018 IEEE Region 10 Conference* (Oct 2018), p. 0038–0043.

[96] Wang, S., Minku, L. L., Ghezzi, D., Caltabiano, D., Tino, P., Yao, X. Concept drift detection for online class imbalance learning. In *The 2013 Int. Joint Conference on Neural Networks (IJCNN)* (Aug 2013), p. 1–10.

[97] Wang, W., Ferrell, B., Xu, X., Hamlen, K. W., Hao, S. Seismic: Secure in-lined script monitors for interrupting cryptojacks. In *European Symposium on Research in Computer Security* (2018), Springer, p. 122–142.

[98] Warneke, D., Kao, O. Nephele: efficient parallel data processing in the cloud. In *Proceedings of the 2nd workshop on many-task computing on grids and supercomputers* (2009), ACM, p. 8.

[99] Widom, J. The starburst rule system: Language design, implementation, and applications. *IEEE Data Engineering Bulletin* (December 1992).

[100] Xin, R. S., Rosen, J., Zaharia, M., Franklin, M. J., Shenker, S., Stoica, I. Shark: SQL and rich analytics at scale. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of data* (2013), ACM, p. 13–24.

[101] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., Stoica, I. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation* (2012), USENIX Association, p. 2–2.

[102] Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., Stoica, I. Discretized streams: Fault-tolerant streaming computation at scale. In *XXIV ACM Symposium on Operating Systems Principles* (2013), ACM, p. 423–438.

[103] Zaharia, M., Das, T., Li, H., Shenker, S., Stoica, I. Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing* (2012), 10–10.

[104] Zhao, S., Chandrashekar, M., Lee, Y., Medhi, D. Real-time network anomaly detection system using machine learning. In *2015 11th International Conference on the Design of Reliable Communication Networks (DRCN)* (March 2015), p. 267–270.