Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Classificação de fluxos de rede IoT MQTT utilizando o EFC

João Victor Bohrer Munhoz

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientador
Prof. Dr. Marcelo Antonio Marotta

Brasília
2022

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Classificação de fluxos de rede IoT MQTT utilizando o EFC

João Victor Bohrer Munhoz

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof. Dr. Marcelo Antonio Marotta (Orientador)
CIC/UnB

Célia Ghedini Ralha
CIC/UnB

João José Costa Gondim
CIC/UnB

Prof. Dr. Marcelo Grandi Mandelli
Coordenador do Bacharelado em Ciência da Computação

Brasília, 26 de Abril de 2022

Dedicatória

Dedico este trabalho a todos os meus colegas da UnB que passei esses últimos semestres de pandemia sem ver. Desejo a todos os meus colegas que se formaram durante esse período boa sorte em suas vidas profissionais e a todos os meus colegas que ainda retornarão à UnB próximo semestre, até logo!

Summer has come and passed
The innocent can never last
Wake me up when September ends

Green Day, 'Wake me up when september ends', 2004

Agradecimentos

Agradeço ao meu orientador, o professor Marcelo Marotta (CIC/UnB), por ter disponibilizado seu tempo para me ajudar a desenvolver este trabalho. À minha colega Manuela Souza pela ajuda com a utilização do EFC e pela paciência com minhas incessantes perguntas. A todos os amigos que me ajudaram e ouviram meus desabafos. A todos os meus familiares que me incentivaram e cuidaram de mim durante essa jornada.

Resumo

Neste trabalho, é proposta a utilização do *Energy-based Flow Classifier (EFC)* para classificar fluxos de redes *Internet of Things (IoT) Message Queuing Telemetry Transport (MQTT)*. Nos últimos anos, vários tipos de classificadores foram propostos para uso em *Network Intrusion Detection Systems (NIDSs)* para IoT, utilizando tanto algoritmos clássicos de inteligência artificial quanto sistemas híbridos que misturam diferentes classificadores. O primeiro tipo mostra bom desempenho, mas não possui a capacidade de classificar fluxos de ataque desconhecidos, enquanto o segundo mostra bons resultados e consegue classificar estes fluxos, mas possui custo computacional elevado para ser utilizado em um contexto IoT, cuja capacidade computacional é limitada. A utilização do EFC vem, portanto, para oferecer uma alternativa viável, visto que possui complexidade de treinamento baixa e é capaz de classificar fluxos de ataques desconhecidos. Para testar seu desempenho, são realizados três experimentos. No primeiro, testa-se a versão binária do EFC para saber se ele é capaz de diferenciar fluxos benignos de fluxos maliciosos. No segundo experimento é utilizada a versão multi-classe do EFC para saber se ele consegue diferenciar os diferentes tipos de fluxo de ataque. Já no último experimento, retirando cada um dos tipos de fluxo de ataque do conjunto de teste, são simulados ataques desconhecidos para testar a eficácia do algoritmo em reconhecer este tipo de fluxo. O EFC mostrou ótimos resultados nos dois primeiros experimentos. Já no terceiro experimento, em dois dos três cenários de teste o EFC mostrou o melhor resultado ou comparável ao melhor, enquanto no cenário restante mostrou resultados abaixo da média, provavelmente por causa do desbalanceamento do *dataset*. Trabalhos futuros podem encontrar mais *datasets* de fluxos IoT MQTT para realizar testes inter-domínio e explorar a possibilidade de realizar *feature selecting* sobre os fluxos.

Palavras-chave: IoT, Aprendizado de Máquina, EFC

Abstract

In this work, the Energy-based Flow Classifier (EFC) algorithm is proposed to classify flows from IoT MQTT networks. Lately, numerous types of classifiers have been proposed for use in Network Intrusion Detection Systems (NIDSs) for IoT networks, ranging from classic AI algorithms to hybrid systems that mix different classifiers. The former type of classifier has shown good results but doesn't have the capacity to classify unknown attack flows, while the latter has also shown good results and is able to classify these flows, but its computational cost is too steep to be used on a IoT context, which often means scarce resources. The use of EFC, therefore, comes as a viable alternative to either of these approaches since its computational cost is low and is able to classify unknown attack flows. To test its efficiency, three experiments were performed: firstly, one-class EFC is tested to confirm that the algorithm is able to distinguish benign flows from malicious ones; next, the multi-class version of EFC is tested to know if it is able to differentiate the types of attack flows; and lastly, removing one type of attack flow from the test set, unknown attacks are simulated to verify the efficiency of the algorithm in classifying this type of flow. Future work should include finding more IoT MQTT datasets to perform inter-domain tests, while also exploring the possibility of doing feature selecting over the flows.

Keywords: IoT, Machine Learning, EFC

Sumário

1 Introdução **1**

2 Fundamentação Teórica **3**

2.1 Energy-based Flow Classifier 3

2.1.1 EFC Multi-classe 5

2.2 O protocolo MQTT 6

2.3 O *dataset* MQTT-IoT-IDS2020 8

3 Trabalhos Relacionados **10**

3.1 Algoritmos de aprendizado de máquina supervisionado tradicionais em fluxos de rede IoT 10

3.2 EFC 12

3.3 O Dataset 14

4 Metodologia **15**

4.1 Organização do *Dataset* 15

4.2 Algoritmos de Classificação 17

4.2.1 Gaussian Naive Bayes 17

4.2.2 Decision Tree 17

4.2.3 Random Forest 18

4.2.4 Multilayer Perceptron 18

4.2.5 Complexidade dos Algoritmos 18

4.3 Desenho dos Experimentos 19

4.3.1 Classificação Binária 20

4.3.2 Classificação Multi-classe 22

4.3.3 Classificação Multi-classe com ataques desconhecidos 23

5 Resultados **24**

5.1 Classificação Binária 24

5.1.1 Utilizando *cutoff_quantile* de 0.95 24

5.1.2 Utilizando *cutoff_quantile* de 0.99 27
5.2 Classificação Multi-classe 30
5.3 Classificação Multi-classe com ataques desconhecidos 35
5.3.1 Fluxos *Aggressive Scan* como ataques desconhecidos 35
5.3.2 Fluxos *MQTT Brute-Force* como ataques desconhecidos 40
5.3.3 Fluxos *Sparta SSH Brute-Force* como ataques desconhecidos 45
5.3.4 Sintetização dos Resultados 50

6 Conclusão **53**

Referências **55**

Apêndice **58**

A Features Extraídas do Dataset **59**

Lista de Figuras

2.1 Grafo que representa o Modelo de Potts para *spins* interagindo em uma estrutura cristalina. 4
2.2 Exemplo de um grafo utilizado pelo EFC 5
2.3 Fase de treinamento do EFC multi-classe. 5
2.4 Fase de teste do EFC multi-classe. 6
2.5 Arquitetura base do protocolo MQTT. 7
2.6 Arquitetura da rede MQTT simulada no *dataset*. 8

4.1 Matriz de confusão. 21

5.1 Gráfico das energias do EFC no cenário de ataque Aggressive Scan 25
5.2 Matriz de confusão do EFC no cenário de ataque Aggressive Scan 25
5.3 Gráfico das energias do EFC no cenário de ataque MQTT Brute-Force . . 26
5.4 Matriz de confusão do EFC no cenário de ataque MQTT Brute-Force . . . 26
5.5 Gráfico das energias do EFC no cenário de ataque Sparta SSH Brute-Force 27
5.6 Matriz de confusão do EFC no cenário de ataque Sparta SSH Brute-Force . 27
5.7 Gráfico das energias do EFC no cenário de ataque Aggressive Scan 28
5.8 Matriz de confusão do EFC no cenário de ataque Aggressive Scan 28
5.9 Gráfico das energias do EFC no cenário de ataque MQTT Brute-Force . . 29
5.10 Matriz de confusão do EFC no cenário de ataque MQTT Brute-Force . . . 29
5.11 Gráfico das energias do EFC no cenário de ataque Sparta SSH Brute-Force 30
5.12 Matriz de confusão do EFC no cenário de ataque Sparta SSH Brute-Force . 30
5.13 Matriz de confusão do *Decision Tree*. 31
5.14 Matriz de confusão do EFC. 32
5.15 Matriz de confusão do *Gaussian Naive Bayes*. 33
5.16 Matriz de confusão do *Multilayer Perceptron*. 34
5.17 Matriz de confusão do *Random Forest*. 35
5.18 Gráfico de distribuição de classes do *Decision Tree*. 36
5.19 Gráfico de distribuição de classes do EFC. 37
5.20 Gráfico de distribuição de classes do *Gaussian Naive Bayes*. 38

5.21 Gráfico de distribuição de classes do *Multilayer Perceptron*. 39
5.22 Gráfico de distribuição de classes do *Random Forest*. 40
5.23 Gráfico de distribuição de classes do *Decision Tree*. 41
5.24 Gráfico de distribuição de classes do EFC. 42
5.25 Gráfico de distribuição de classes do *Gaussian Naive Bayes*. 43
5.26 Gráfico de distribuição de classes do *Multilayer Perceptron*. 44
5.27 Gráfico de distribuição de classes do *Random Forest*. 45
5.28 Gráfico de distribuição de classes do *Decision Tree*. 46
5.29 Gráfico de distribuição de classes do EFC. 47
5.30 Gráfico de distribuição de classes do *Gaussian Naive Bayes*. 48
5.31 Gráfico de distribuição de classes do *Multilayer Perceptron*. 49
5.32 Gráfico de distribuição de classes do *Random Forest*. 50

Lista de Tabelas

4.1 Cenário no qual a rede não sofreu nenhum tipo de ataque 16
4.2 Cenário no qual a rede sofreu ataques do tipo Aggressive Scan 16
4.3 Cenário no qual a rede sofreu ataques do tipo UDP Scan 16
4.4 Cenário no qual a rede sofreu ataques tipo MQTT Brute-Force 16
4.5 Cenário no qual a rede sofreu ataques do tipo Sparta SSH Brute-Force . . 16
4.6 Tabela de complexidade de treinamento dos algoritmos 19

5.1 Métricas de desempenho do EFC no cenário de ataque Aggressive Scan . . 24
5.2 Métricas de desempenho do EFC no cenário de ataque MQTT Brute-Force 25
5.3 Métricas de desempenho do EFC no cenário de ataque Sparta SSH Brute-Force 26
5.4 Métricas de desempenho do EFC no cenário de ataque Aggressive Scan . . 27
5.5 Métricas de desempenho do EFC no cenário de ataque MQTT Brute-Force 28
5.6 Métricas de desempenho do EFC no cenário de ataque Sparta SSH Brute-Force 29
5.7 Métricas de desempenho do *Decision Tree* 30
5.8 Métricas de desempenho do EFC Multi-classe 31
5.9 Métricas de desempenho do *Gaussian Naive Bayes* 32
5.10 Métricas de desempenho do *Multilayer Perceptron* 33
5.11 Métricas de desempenho do *Random Forest* 34
5.12 Métricas de desempenho do *Decision Tree* 36
5.13 Métricas de desempenho do EFC 36
5.14 Métricas de desempenho do *Gaussian Naive Bayes* 38
5.15 Métricas de desempenho do *Multilayer Perceptron* 39
5.16 Métricas de desempenho do *Random Forest* 40
5.17 Métricas de desempenho do *Decision Tree* 41
5.18 Métricas de desempenho do EFC 42
5.19 Métricas de desempenho do *Gaussian Naive Bayes* 43
5.20 Métricas de desempenho do *Multilayer Perceptron* 44
5.21 Métricas de desempenho do *Random Forest* 45

5.22 Métricas de desempenho do *Decision Tree* 46
5.23 Métricas de desempenho do EFC 47
5.24 Métricas de desempenho do *Gaussian Naive Bayes* 48
5.25 Métricas de desempenho do *Multilayer Perceptron* 49
5.26 Métricas de desempenho do *Random Forest* 50
5.27 Tabela de comparação da média ponderada do f1-score de cada algoritmo. 51

Lista de Abreviaturas e Siglas

ANN Artificial Neural Network.

CSV Comma-separated Values.

DS2OS Distributed Smart Space Orchestration System.

EFC Energy-based Flow Classifier.

IoT Internet of Things.

KNN K-Nearest Neighbors.

ML Machine Learning.

MQTT Message Queuing Telemetry Transport.

NIDS Network Intrusion Detection System.

NIDSs Network Intrusion Detection Systems.

pcap Packet Capture.

SVM Support Vector Machine.

Capítulo 1

Introdução

De acordo com a empresa de segurança cibernética Kaspersky [1], na primeira metade de 2021, o número de ataques a seus *honeypots* que simulam dispositivos *Internet of Things (IoT)* desprotegidos aumentou em cerca de 137% se comparado à segunda metade de 2020. Esses ataques cibernéticos podem ser responsáveis por uma diversa gama de problemas; do roubo de informações pessoais à extorsão de milhões de dólares via *ransomware* [2] [3] [4]. É imperativo, portanto, que haja modos de proteger essas redes IoT contra tais ataques.

A capacidade de processamento e de armazenamento de redes IoT é geralmente limitada pelos recursos disponíveis, como tamanho dos dispositivos, consumo de energia e capacidade computacional [5]. Estas características fazem com que algoritmos complexos que exijam muita memória ou espaço de armazenamento, por exemplo, não sejam aplicáveis. São necessárias, então, soluções com requisitos computacionais reduzidos e que impactem minimamente o desempenho da rede. Nesse contexto, *Network Intrusion Detection Systems (NIDSs)* baseados na análise de fluxos da rede são uma ferramenta valiosa para classificar e lidar com diferentes ameaças em redes IoT. Por analisarem e classificarem fluxos de rede, ou seja, coleções de pacotes, e não cada pacote individualmente, possuem um impacto menor no desempenho da rede.

Nos últimos anos, vários tipos de classificadores foram propostos para uso em fluxos IoT, com destaque para a utilização de algoritmos de aprendizado de máquina supervisionado tradicionais [6] [7] como *Random Forest* e *Decision Tree*, e sistemas híbridos misturando diferentes tipos de classificadores [8] [9]. O primeiro tipo mostra uma excelente taxa de acerto e uma baixa taxa de falsos positivos, enquanto o segundo mantém essas boas faixas, além de possuir a capacidade de detectar ataques desconhecidos, uma característica muito desejada. Entretanto, o encadeamento de técnicas necessárias para a implementação desses classificadores híbridos faz com que seus custos computacionais aumentem consideravelmente, possivelmente sobrecarregando redes com recursos dispo-

níveis escassos. É desejável, portanto, um algoritmo de classificação que seja capaz de identificar ataques conhecidos e detectar ataques desconhecidos em um único passo, com custo computacional que seja comportado pelos escassos recursos de redes IoT.

Utilizando um *dataset* [10] de fluxo de dados de uma rede IoT com diferentes ataques como base para os resultados deste trabalho, propõe-se a utilização do classificador Energy-based Flow Classifier (EFC), introduzido inicialmente por Pontes et al. [11] [12] e mais tarde expandido em [13]. Esse classificador possui as duas qualidades ressaltadas no parágrafo anterior, além de uma complexidade temporal menor do que a dos classificadores baseados em algoritmos clássicos de inteligência artificial [11] [14]. Contudo, os *datasets* [15] [16] em que o EFC mostrou ter essas qualidades não eram de fluxos de rede IoT. Então, ao testá-lo nesse novo *dataset*, busca-se saber se o EFC é adequado também para a utilização em NIDSs no contexto de redes IoT.

O restante do trabalho está disposto da seguinte maneira: o Capítulo 2 trará um embasamento teórico sobre os principais componentes desse artigo, com ênfase em como o EFC e o protocolo Message Queuing Telemetry Transport (MQTT) do *dataset* funcionam;, o Capítulo 3 discorrerá sobre os trabalhos relacionados e suas principais contribuições; o Capítulo 4 apresentará a metodologia dos experimentos; o Capítulo 5 apresentará e discutirá os resultados dos experimentos; e por fim o Capítulo 6 apresentará as conclusões e as sugestões para trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Este capítulo introduzirá os tópicos necessários para que possa haver um melhor entendimento do assunto apresentado nesse trabalho, sendo dividido em três seções. Na Seção 2.1, será abordado como funciona o EFC, o classificador cujo desempenho está sendo analisado no trabalho. Na Seção 2.2, será abordado o protocolo de rede MQTT e suas principais características. Já na Seção 2.3, será apresentado o *dataset* MQTT-IoT-IDS2020, utilizado nesse trabalho para avaliar o desempenho do EFC, mostrando quais tipos de dispositivo de rede foram utilizados e os diversos cenários simulados.

2.1 Energy-based Flow Classifier

O EFC é um algoritmo de estatística inversa, ou seja, infere uma distribuição estatística global baseada em uma pequena amostra dessa distribuição [17]. Ele é fundamentado no modelo de Potts [18], que faz uma descrição matemática de um conjunto de spins interagindo em uma estrutura cristalina. Nesse modelo, o conjunto de *spins* interagindo é mapeado em um grafo conexo $G(\eta, \epsilon)$, apresentado na Figura 2.1, onde cada nó $i \in \eta = \{1, ..., N\}$ é associado a um *spin* a_i, que pode assumir valores de um conjunto Ω que contém todos os estados quânticos possíveis para ele. Cada nó i também possui um campo local $h_i(a_i)$ associado a ele, que é uma função do estado de a_i. Além disso, cada aresta (i, j) possui um valor de acoplamento $e_{ij}(a_i, a_j)$ que é a função dos estados dos spins a_i e a_j associados ao nós i e j. Cada configuração específica de um sistema de *spins* possui uma energia total associada, determinada pela função Hamiltoniana $\mathcal{H}(a_1...a_n)$, que depende do estado de todos os *spins*.

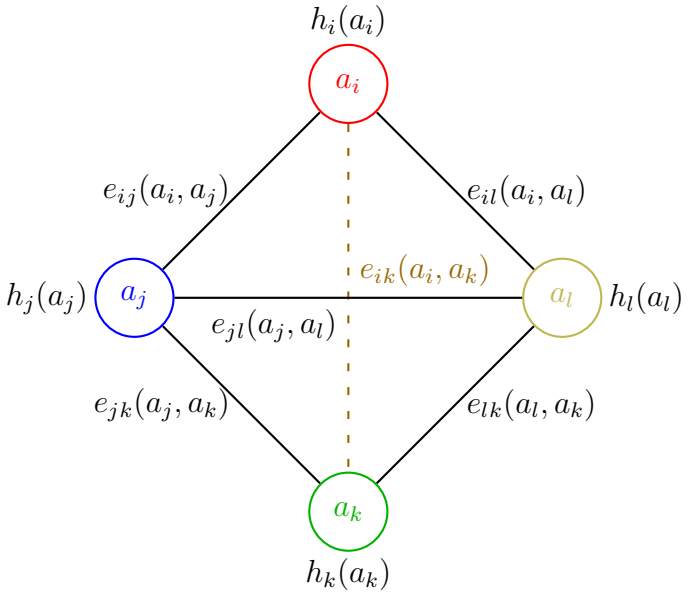

Figura 2.1: Grafo que representa o Modelo de Potts para *spins* interagindo em uma estrutura cristalina.

No EFC, as intuições do modelo de Potts são reutilizadas para representar fluxos de rede [11]. Um fluxo k qualquer é representado por um grafo do tipo $G_k(\eta, \epsilon)$. Em vez de *spins*, cada nó do grafo representa uma *feature* específica $i \in \eta = \{SrcPort, ..., Flags\}$. Nestes nós, cada *feature* i assume um valor a_{ki} de um conjunto Ω_i que contém todos os possíveis valores para essa *feature*. O modelo está exemplificado na figura Figura 2.2, onde dentro de cada nó está o valor da *feature*, ao lado deles o campo local e nas arestas o valor de acoplamento

Uma vez que os valores dos campos locais e acoplamentos dependem dos valores assumidos pelas *features* em um dado fluxo, cada fluxo possui uma combinação única desses valores. Assim como no modelo de Potts, a Hamiltoniana dos campos locais e acoplamentos determina a "energia" total $\mathcal{H}(a_{k1}...a_{kn})$ de cada fluxo, que é dada por:

$$\mathcal{H}(a_{k1}...a_{kn}) = -\sum_{i,j|i<j} e_{ij}(a_{ki}, a_{kj}) - \sum_i h_i(a_{ki}) \tag{2.1}$$

Como o EFC é um classificador baseado em anomalias, o modelo estatístico utilizado para classificação é inferido baseado somente em fluxos benignos [11]. Espera-se, portanto, que amostras de fluxos benignos possuam energia mais baixa do que de fluxos maliciosos. Em outras palavras, o valor da energia de um fluxo é o quão dissimilar ele é do conjunto de fluxos utilizados para inferir o modelo na fase de treinamento. Desse modo, é possível classificar amostras de fluxos como benignas ou maliciosas baseado em um valor limite de energia. Fluxos com energia menor que o limite são classificados como benignos e os com energia maior são classificados como maliciosos.

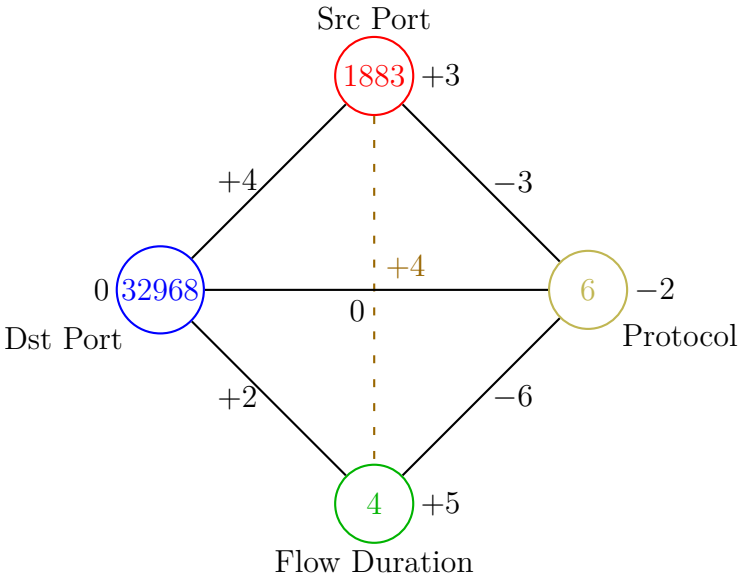

Figura 2.2: Exemplo de um grafo utilizado pelo EFC

2.1.1 EFC Multi-classe

O EFC multi-classe utiliza os mesmos princípios do EFC binário, decidindo se um fluxo pertence a uma certa classe ou não comparando o valor de sua energia. Entretanto, na versão binária do algoritmo a energia dos fluxos é calculada utilizando somente uma única classe como referência para o modelo estatístico durante o treinamento. Já no multi-classe, vários modelos estatísticos são inferidos, um para cada classe [13].

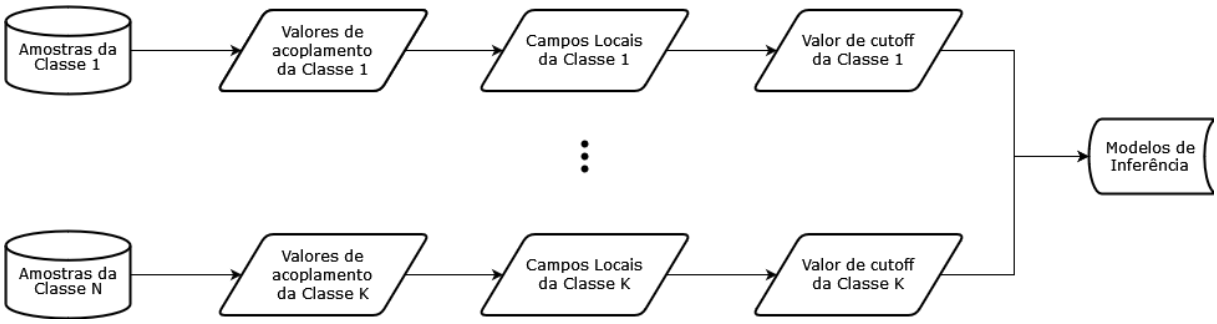

Figura 2.3: Fase de treinamento do EFC multi-classe.

A Figura 2.3 ilustra o processo de treinamento do EFC multi-classe. Esta fase consiste na replicação do processo de treinamento do EFC binário para cada uma das classes presentes no conjunto de treinamento, e não somente nas classes de fluxos benignos. Os modelos são então inferidos e os limites de energia calculados para cada classe, da mesma forma que ocorre no EFC binário. Por fim, os modelos inferidos são armazenados para serem utilizados na fase de classificação.

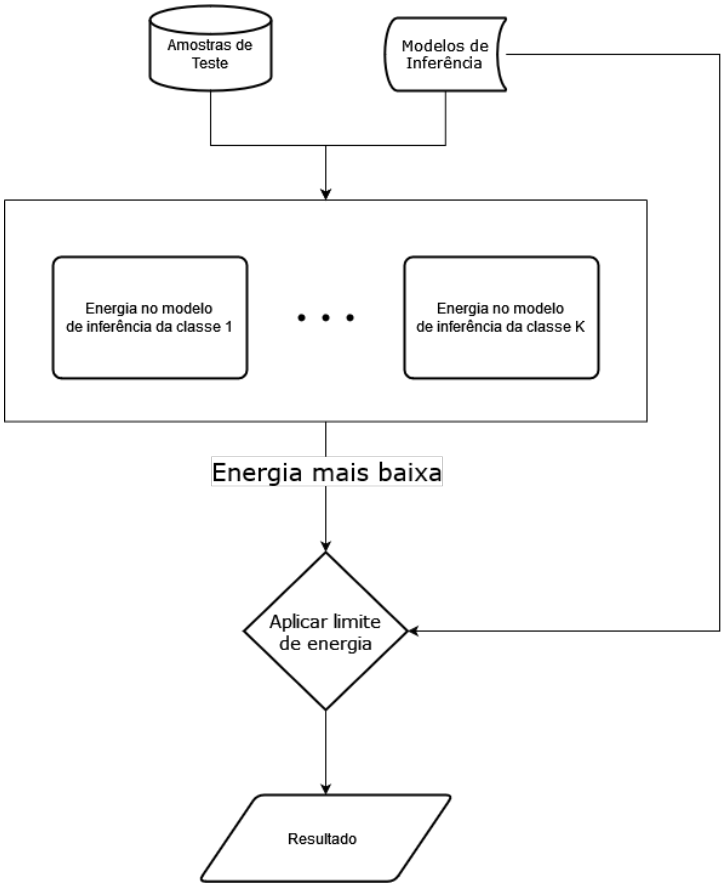

Figura 2.4: Fase de teste do EFC multi-classe.

Já a Figura 2.4 ilustra o processo de teste do EFC multi-classe. Para classificar um fluxo do conjunto de teste, sua energia é calculada para cada modelo inferido durante a etapa de treinamento, gerando um vetor de energias para cada fluxo. Depois de calcular estas energias, o EFC pega o menor valor gerado e compara com o limite de energia daquela classe, uma vez que a menor energia representa uma maior similaridade entre os fluxos. Se a energia está abaixo do limite, o fluxo é considerado como sendo da classe que gerou a energia, de outro modo, ele é classificado como *suspeito*. Esta segunda situação significa que até mesmo a classe que mais se assemelha a esse fluxo não é similar o suficiente. Ela é, portanto, classificada como suspeita, provavelmente se tratando de um fluxo de ataque desconhecido [13].

2.2 O protocolo MQTT

O MQTT é um protocolo de rede com um *code footprint* pequeno e uso mínimo de banda do tipo *publish-subscribe*, feito para transportar mensagens entre dispositivos [19]. Nesta

organização, os dispositivos que mandam mensagens, chamados *publishers*, não programam as mensagens para chegar diretamente aos dispositivos que recebem as mensagens, chamados *subscribers*. Em vez disso, categorizam as mensagens que mandam em tópicos, sem o conhecimento se sequer existe algum *subscriber*. Do mesmo modo, os *subscribers* expressam interesse em um ou mais tópicos e recebem somente mensagens desses tópicos, sem o conhecimento de quem é o *publisher* [20].

O protocolo MQTT define dois tipos de entidade de rede: um *broker* e clientes. Um *broker* MQTT é um servidor que recebe todas as mensagens dos clientes e então as encaminha para os clientes de destino. Já um cliente MQTT pode ser qualquer dispositivo que acessa uma biblioteca MQTT para executar um programa e se conecta ao *broker* através da rede [21].

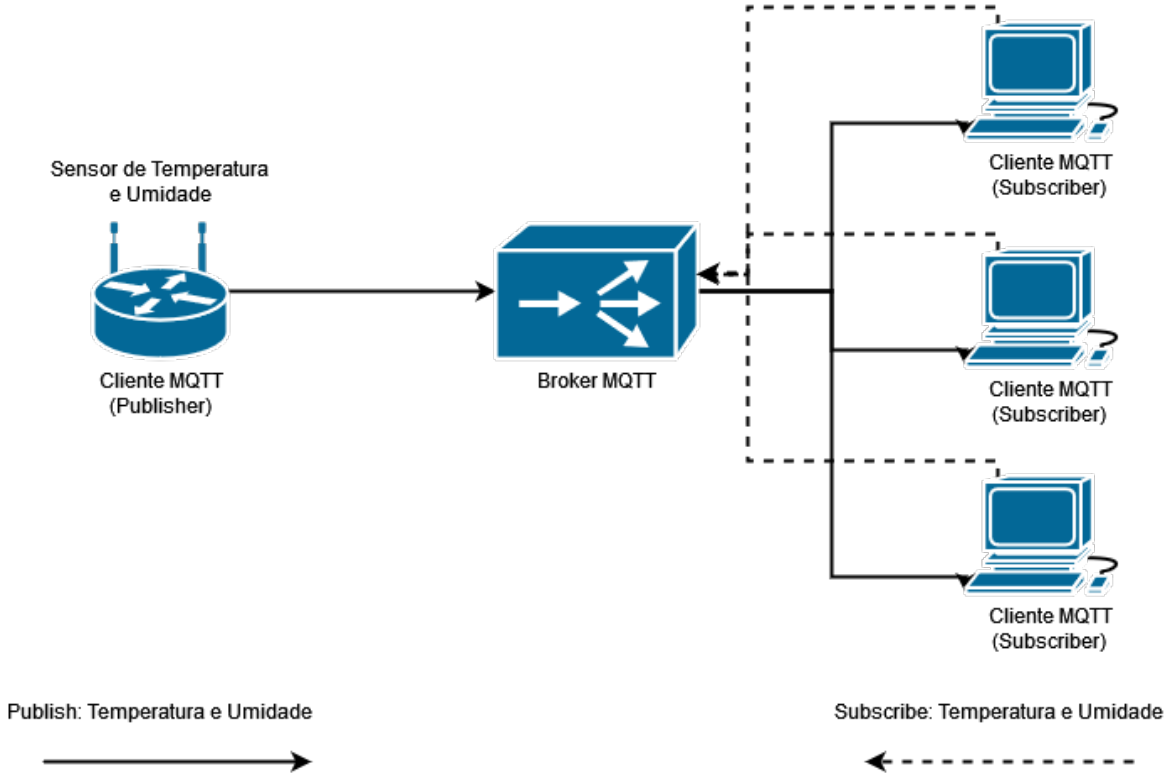

Figura 2.5: Arquitetura base do protocolo MQTT.

Um cliente MQTT pode agir como um *publisher* para mandar informações ao *broker*. No exemplo ilustrado na Figura 2.5, o cliente MQTT à esquerda é um sensor que publica informações de temperatura e umidade relativa, denotados por *Temp* e *RH*, respectivamente. Já a aplicação à direta é um cliente MQTT que é *subscriber* das informações de *Temp*/*RH* sendo geradas pelos sensores.

A presença do *broker* MQTT faz com que os *publishers* não precisem lidar com a transmissão de suas informações diretamente para cada um dos *subscribers*. Um dos benefícios

dessa dissociação é que o *broker* consegue garantir o armazenamento das informações recebidas em *cache* para casos de falhas de rede. Além disso, torna desnecessário que tanto o *publisher* quanto o *subscriber* tenham que estar *online* ao mesmo tempo para que possa haver recebimento de mensagens.

2.3 O *dataset* MQTT-IoT-IDS2020

Publicado por Hindy et al. [10], o *dataset* utilizado neste trabalho é uma rede MQTT simulada. Como ilustrado na Figura 2.6, esta rede é composta por doze sensores MQTT, um *broker*, um servidor que simula a transmissão de uma câmera e outro que simula um invasor, todos eles simulados utilizando máquinas virtuais.

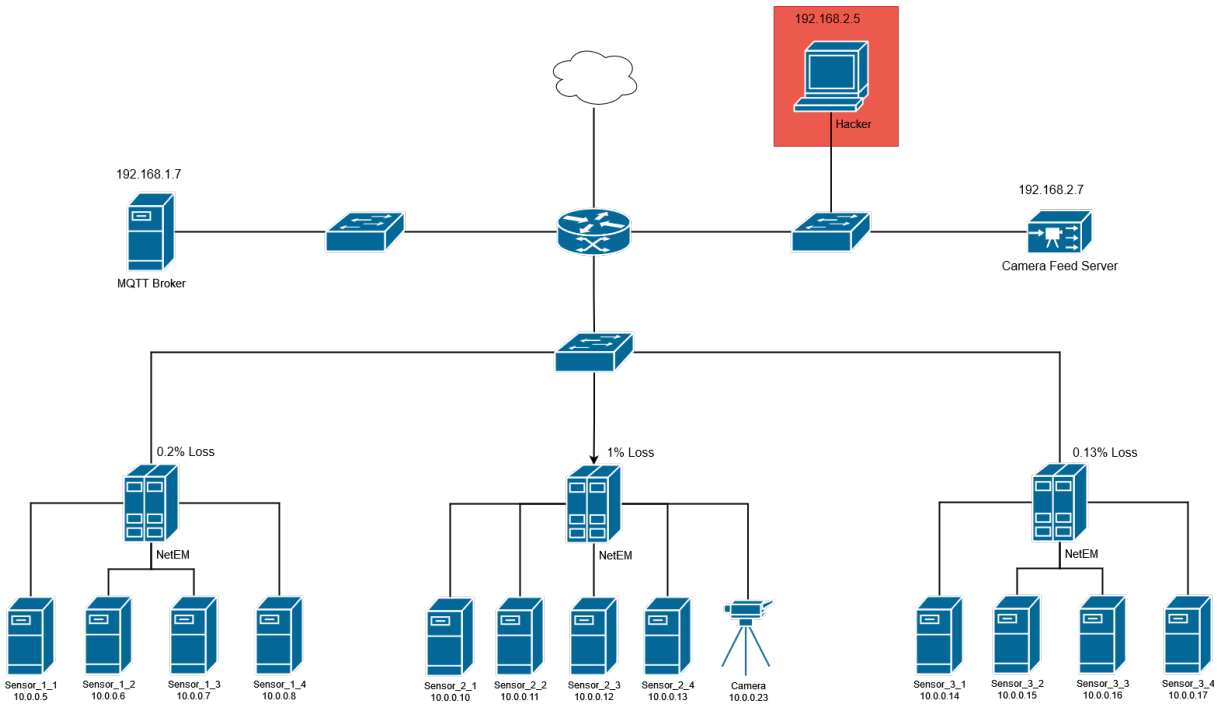

Figura 2.6: Arquitetura da rede MQTT simulada no *dataset*.

Durante operação normal, ou seja, quando não estão sofrendo nenhum tipo de ataque, todos os doze sensores mandam mensagens com conteúdo aleatório utilizando o comando *Publish* do protocolo MQTT, sendo que o tamanho das mensagens pode diferir entre sensores para simular diferentes cenários de uso. A transmissão da câmera é simulada através de um *stream* UDP do *VLC Media Player* e para simular um cenário ainda mais realista alguns pacotes são descartados pelos emuladores de rede, cada um deles descartando respectivamente 0.2%, 1% e 0.13% dos pacotes.

O *dataset* fornece cinco diferentes cenários de simulação, sendo um deles de funcionamento normal contendo somente fluxos benignos e os outros quatro de cenários de ataque contendo tanto fluxos benignos quanto fluxos maliciosos. Todos os cenários foram criados utilizando o programa tcpdump [22] para gravar o tráfego de rede e exportar para arquivos pcap.

Os ataques presentes no *dataset* são:

- *Aggressive Scan*
- *UDP Scan*
- *Sparta SSH Brute-Force*
- *MQTT Brute-Force*

Tanto o *Aggressive Scan* quanto o *UDP Scan* são meramente tipos de *network scanning* [23], uma técnica que embora não seja inerentemente perigosa — podendo ser inclusive utilizada por sistemas de monitoramento e segurança [24] — permite que pessoas com intenções maliciosas descubram informações sobre a rede atacada. Com estas informações é possível explorar vulnerabilidades e comprometer a segurança da rede [25]. No contexto do *dataset*, ambos foram simulados utilizando o programa Nmap [26].

Já o *Sparta SSH Brute-Force* e o *MQTT Brute-Force* caem na categoria de ataques *brute-force* [27], o que geralmente envolve inúmeras tentativas por parte do agente malicioso de adivinhar as credenciais da rede para poder ter total acesso a ela [28]. No contexto do *dataset*, os ataques *Sparta SSH Brute-Force* foram simulados utilizando a ferramenta Sparta [29] e os ataques *MQTT Brute-Force* utilizando a ferramenta MQTT-PWN [30].

Capítulo 3

Trabalhos Relacionados

Neste capítulo são mencionados dois artigos [7] [6] nos quais foram analisados os desempenhos de classificadores baseados em algoritmos de aprendizado de máquina supervisionado tradicionais, sendo eles respectivamente testados em um *dataset* binário e multi-classe. Em seguida, são analisados dois artigos sobre o classificador EFC que será utilizado para os testes aqui, sendo o primeiro artigo aquele que inicialmente o propôs como um classificador binário [11] e o segundo como um classificador multi-classe [13]. E por fim, é mencionado o artigo [10] no qual foi proposto o *dataset* a ser utilizado para os testes neste trabalho, baseado no protocolo MQTT com fluxos normais e quatro diferentes tipos de ataque.

3.1 Algoritmos de aprendizado de máquina supervisionado tradicionais em fluxos de rede IoT

Em Khraisat et al. [7], procurou-se saber se algoritmos de Machine Learning (ML) tradicionais eram capaz de diferenciar fluxos benignos de fluxos maliciosos em um contexto de rede IoT. Foram utilizados os algoritmos *Support Vector Machine (SVM)*, *Naive Bayes*, *Decision Tree* e *Adaboost*, dois dos quais também serão utilizados neste trabalho. Outro ponto que vale notar é que o *dataset* em que foram realizados os testes foi proposto e criado pelos autores, nominado de *Sensor480*. Esse *dataset* é assim batizado pois possui 480 fluxos com classes binárias, com *Normal* representando fluxos benignos e *Attack* representando fluxos maliciosos.

Para construir o sistema utilizado para a criação do *dataset*, foram utilizados uma placa Arduino específica para ambientes IoT que se comunicará com uma plataforma *ThinkSpeak* na nuvem, um sensor de umidade e temperatura que fornecerá os dados a serem enviados e um *laptop* que realizará o papel de agente malicioso. Para simular esse agente malicioso,

serão utilizadas ferramentas nativas do Kali Linux para simular ataques à rede, uma das quais também é utilizada para simular um dos ataques no *dataset* utilizado neste trabalho. Os fluxos normais são simulados deixando a comunicação ocorrer normalmente com a plataforma *ThinkSpeak*. Já os fluxos maliciosos são simulados utilizando as ferramentas *Ettercap* e *Burp Suite* para interceptar os pacotes e modificar seus valores, para só então mandá-los para a plataforma *ThinkSpeak*. Utilizando a ferramenta *Wire shark*, esses fluxos foram capturados, classificados e salvos em formato Comma-separated Values (CSV), o qual formará o *dataset* utilizado para os testes.

Com base nesse *dataset* criado, foram realizados os testes com os algoritmos citados anteriormente. Analisando os resultados, o algoritmo com melhor desempenho foi o *Decision Tree*, uma vez que conseguiu classificar perfeitamente todos os fluxos, tendo em vista seus valores de sensibilidade (*recall*), precisão (*precision*), f1-escore (*f1-score*) e acurácia (*accuracy*) de 1.00. Os outros algoritmos também realizaram um ótimo trabalho, uma vez que nenhuma das métricas dos algoritmos testados ficou abaixo de 0.96. Embora os resultados sejam encorajadores, nesse trabalho não foi testado o desempenho dos algoritmos em fluxos de rede IoT com mais de uma classe de ataque, o que em NIDSs atualmente é um pré-requisito. Portanto, para o escopo deste trabalho, buscou-se algoritmos que fossem testados em *datasets* IoT que não fossem estritamente binários que nem o *Sensor480*.

Em Hasan et al. [6], procurou-se saber se algoritmos de aprendizado de máquina supervisionado tradicionais eram eficientes para classificar diferentes ataques a redes IoT. Nele, foram utilizados os algoritmos *Logistic Regression*, SVM, *Decision Tree*, *Random Forest* e Artificial Neural Network (ANN), muitos dos quais também serão utilizados neste trabalho. Para testar a eficácia desses algoritmos foi usado o *dataset DS2OS Traffic Traces* [31], que simula fluxos de rede IoT em um ambiente Distributed Smart Space Orchestration System (DS2OS). Uma vez com os dados do *dataset*, é feito o pré-processamento, o que inclui limpeza de dados e conversão de dados categóricos.

O *dataset* é então separado em cinco conjuntos de treinamento e teste para que possa ser realizada uma referência cruzada com cinco grupos de amostras. Feito isso, os algoritmos são executados sobre os conjuntos, retornando um modelo preditivo nos quais ainda são aplicadas técnicas de otimização. No modelo gerado pelo algoritmo *Logistic Regression* foi aplicado coordenada descendente, sobre os do SVM e ANN foi aplicado gradiente descendente e sobre do *Decision Tree* e *Random Forest* não foi aplicado nenhum algoritmo de otimização por se tratarem de modelos não-paramétricos. Com os modelos gerados e otimizados, são por fim realizados as predições sobre o conjunto de teste.

Os resultados mostram que o *Random Forest* obteve o melhor desempenho. Em todas as métricas apresentadas pelo trabalho, o que inclui *precision*, *recall*, *accuracy* e *f1-score*, o algoritmo estava sempre entre os melhores. É válido notar, entretanto, que a maioria dos

algoritmos utilizados nesse trabalho obtiveram bom resultado, com destaque para SVM e ANN. Estes algoritmos tradicionais, entretanto, não são capazes de classificar ataques desconhecidos, o que hoje em dia é uma característica muito desejável em NIDSs. Desta forma, para o escopo deste artigo, buscou-se outro algoritmo que identificasse ataques desconhecidos.

3.2 EFC

Em Pontes et al. [11], foi proposto o EFC, um classificador baseado no modelo de Potts [18] da mecânica estatística, o qual faz uma descrição matemática de um conjunto de *spins* interagindo em uma estrutura cristalina, relação que pode ser mapeada em um grafo $G(\eta, \epsilon)$. O EFC traz essa relação para o contexto de classificadores, que pode ser mapeada em um grafo $G_k(\eta, \epsilon)$, onde em vez de *spins*, cada nó do grafo representa um atributo (*feature*) específico $i \in \eta = \{SrcPort, ..., Flags\}$. Nestes nós, cada *feature* i assume um valor a_{ki} de um conjunto Ω_i que contém todos os possíveis valores para essa *feature*, sendo k um fluxo qualquer. Os nós e a relação entre eles gera uma "energia"de acordo com a dissimilaridade entre os fluxos em que o algoritmo foi treinado e o fluxo que está sendo avaliado.

Quanto mais baixa for a energia de um fluxo, mais similar ele é ao conjunto de fluxos com que o modelo foi inferido na fase de teste. Isso faz com que o EFC seja ideal para ser utilizado como um classificador baseado em anomalias, uma vez que o modelo inferido neste contexto pode ser treinado somente com fluxos benignos. Isso traz o benefício de que, uma vez que ele é treinado para reconhecer somente fluxos benignos, todos os fluxos maliciosos devem ser rejeitados, incluindo fluxos de ataque desconhecidos, uma das características que se busca neste trabalho. No artigo, além do EFC, são testados os algoritmos *Naive Bayes*, K-Nearest Neighbors (KNN), *Decision Tree*, SVM, *Multilayer Perceptron*, *Adaboost* e *Random Forest* para motivos de comparação. No ambiente de teste, são utilizados os *datasets CIDDS-001*, *CICIDS17* e *CICDDoS19*, os quais oferecem várias classes de ataque e misturam tráfegos de rede simulados e tráfegos de redes reais.

Analisando os resultados, o EFC possui eficiência um pouco menor quando treinado e testado no mesmo *dataset* se comparado ao *Random Forest* e *Decision Tree*, os quais apresentam melhores resultados, mas possui larga vantagem em relação aos outros algoritmos quando treinado em um *dataset* e testado em outro, o que mostra uma maior adaptabilidade a diferentes contextos. Entretanto, trata-se de um classificador binário, ou seja, ele reconhece um fluxo como normal ou malicioso, sem reconhecer o tipo de fluxo malicioso, o que impede o *NIDS* de tomar medidas prontamente contra aquele tipo de ataque específico. Embora ele reconheça ataques desconhecidos, esperamos que no es-

copo desse trabalho o algoritmo também identifique os tipos de ataques conhecidos, o que motivou a procura de outro algoritmo.

Em Pontes et al. [13], foi proposto o EFC Multi-classe, uma evolução do EFC proposto em Pontes et al. [11]. Como seu nome dá a entender, ele é capaz de classificar várias classes ao mesmo tempo, o que significa que, além de conseguir separar fluxos em benignos e maliciosos, consegue identificar os tipos de ataque desses fluxos maliciosos. Para isso, em vez de criar somente um modelo preditivo com base nos fluxos benignos, é criado um modelo para cada classe presente no conjunto de teste.

Para classificar um fluxo, ele tem sua energia calculada por cada um dos modelos preditivos, pertencendo à classe a qual o modelo preditivo retornar menor energia, desde que essa energia seja menor do que o limite de corte (*cutoff*). Caso a menor energia retornada ainda seja maior que esse valor, o fluxo é considerado suspeito, o que geralmente indica que é um ataque desconhecido. Essa é a combinação ideal de características que se procura para o classificador deste trabalho, entretanto, é necessário analisar o desempenho do EFC antes de considerá-lo como um candidato. No artigo, além do EFC, são testados os algoritmos *Naive Bayes*, KNN, *Decision Tree*, *SVC*, *Multilayer Perceptron* e *Random Forest* para que seus resultados possam servir de comparação. Para o ambiente de teste, são utilizados os *datasets CIDDS-001* e *CICIDS2017*, os quais possuem diversos tipos de ataque.

Para os testes somente com classes conhecidas, todos os algoritmos foram testados sobre os dois *datasets* e a partir do resultado de cada um foi calculado seu *f1-score*, que serviu como métrica de desempenho para comparação. Os resultados mostram que, nesse cenário, os algoritmos com melhor desempenho foram o EFC e o *Random Forest*, os quais revezaram as melhores métricas de classificação para diferentes tipos de fluxo, embora o *Random Forest* leve certa vantagem. Tendo em vista que os dois algoritmos foram os melhores classificadores nessa etapa do experimento, o teste com classes desconhecidas foi realizado somente com eles.

Nesses testes, para simular fluxos de ataques desconhecidos, o método aplicado foi retirar sistematicamente um tipo de fluxo de ataque do conjunto de treinamento, mas deixá-lo no conjunto de teste, a mesma técnica utilizada neste trabalho. Os resultados mostram que, nesse cenário, o EFC leva vantagem em relação ao *Random Forest*, o que novamente ilustra a adaptabilidade do algoritmo a contextos de uso real. Entretanto, ele foi testado somente em *datasets* de redes que não eram de IoT. Neste trabalho, portanto, será utilizada essa versão do algoritmo para testar se a sua eficiência se mantém ao utilizar *datasets* que simulam esse tipo de rede.

3.3 O Dataset

Em Hindy et al. [10], foi proposta a criação de um novo *dataset* de fluxo de rede IoT baseado no protocolo MQTT. Esse *dataset* é composto por 12 sensores, um *broker* MQTT, um servidor que simula a transmissão de uma câmera e outro que simula um invasor, todos eles simulados utilizando máquinas virtuais. Além de fluxos normais, o *dataset* simula quatro diferentes tipos de ataque, o *Aggressive Scan*, *UDP Scan*, *Sparta SSH Brute-Force* e *MQTT Brute-Force*, dos quais somente o último é específico para redes MQTT, sendo o resto ataques de rede genéricos.

Todos os cenários disponibilizados no *dataset* incluem fluxos benignos misturados com fluxos maliciosos, de modo a simular um caso de uso mais realista, sendo que em cada cenário há somente um tipo de ataque. Todos esses cenários foram criados utilizando o programa *tcpdump* para gravar o tráfego de rede e exportar para arquivos pcap, um *container* binário que armazena as informações do tráfego de rede gravado. A partir desses arquivos é possível criar CSVs que podem então ser executados por classificadores, o que foi feito no artigo para poder testar a aptidão do *dataset* para ser utilizado em trabalhos que buscam construir e avaliar classificadores para NIDSs, como é o caso deste trabalho.

Os algoritmos utilizados para avaliar o *dataset* foram *Logistic Regression*, *Gaussian Naive Bayes*, KNN, SVM, *Decision Tree* e *Random Forest*. Todos os algoritmos foram testados sobre o *dataset* e a partir do resultado de cada um foram calculados os valores de suas métricas *precision*, *recall* e *f1-score*. Os resultados dos experimentos ressaltaram que os ataques de rede genéricos são facilmente distinguíveis de fluxos benignos devido às suas características próprias diferentes se comparado ao comportamento padrão de uma rede IoT. Já os ataques *MQTT Brute-Force* são mais complicados e podem facilmente imitar um fluxo benigno, uma vez que também utilizam os comandos normais de uma rede MQTT como *publish* e *subscribe*. Esse *dataset* se faz, portanto, ideal para que se possa testar o algoritmo EFC escolhido para o escopo deste trabalho, uma vez que é um *dataset* IoT e possui tanto ataques genéricos quanto ataques específicos para o protocolo sendo utilizado, o que é ideal para o teste de algoritmos que se queira utilizar em um NIDS.

Capítulo 4

Metodologia

Este capítulo é divido em três seções. Na Seção 4.1, será discutida a estrutura dos fluxos de rede do *dataset* utilizado neste trabalho, apontando algumas das *features* presentes, como foram extraídas e qual a proporção de fluxos maliciosos para fluxos benignos. Na Seção 4.2, será feita uma descrição sucinta dos algoritmos de aprendizado de máquina supervisionado tradicionais que foram utilizados para comparar a eficácia do EFC e discutir as suas complexidades. E por fim, na Seção 4.3, será descrito o desenho dos experimentos realizados neste trabalho.

4.1 Organização do *Dataset*

Para este trabalho será utilizado o *dataset* MQTT-IoT-IDS2020 [10], como detalhado na Seção 2.3. Ele fornece os fluxos de rede necessários para que se possa testar o desempenho do EFC. Entretanto, somente os arquivos *pcap* criados pelo *tcpdump* não são suficientes para que os algoritmos consigam analisar os dados, uma vez que este tipo de arquivo está em formato binário. É necessário, portanto, extrair as *features* dos fluxos de rede presentes no arquivo binário e salvá-las em um arquivo CSV.

Em Hindy et al. [10] também são fornecidas versões em CSV dos cenários do *dataset*. Somente com eles já seria possível testar os algoritmos, entretanto foi decidido pela extração própria das *features* através dos arquivos pcap fornecidos. Isso para que a formatação dos arquivos CSV seja sempre a mesma, facilitando inclusive a comparação com outros *datasets* futuramente, desde que os arquivos pcap também sejam fornecidos.

Para realizar este processo, foi utilizada a ferramenta CICFlowMeter [32]. Sua escolha deve-se principalmente ao fato de ser a mesma ferramenta utilizada em Pontes et al. [11] [13] e por ser um software livre e de código aberto. O arquivo CSV gerado por ele possui 84 *features*, as quais podem ser encontradas no Apêndice A ao final do documento.

Antes que estes dados pudessem ser usados pelos algoritmos, foi necessário adicionar manualmente as *labels* em cada fluxo de rede presente no arquivo CSV. Sabe-se por [10] que os fluxos maliciosos do *dataset* são caracterizados exclusivamente pelo IP 192.168.2.5 e, portanto, os fluxos que mandam ou recebem dados envolvendo este IP são caracterizados como fluxos maliciosos, com cada tipo de ataque recebendo um *label* único. E, por consequência, os fluxos que não mandam ou não recebem dados envolvendo este IP são sempre caracterizados como fluxos benignos. Os cenários e a quantidade de fluxos benignos e fluxos maliciosos presentes em cada um estão nas Tabelas 4.1 a 4.5:

Tabela 4.1: Cenário no qual a rede não sofreu nenhum tipo de ataque

Label	**Número de Fluxos**
Benign	167159

Tabela 4.2: Cenário no qual a rede sofreu ataques do tipo Aggressive Scan

Labels	**Número de Fluxos**
Benign	11187
Aggressive Scan	20058

Tabela 4.3: Cenário no qual a rede sofreu ataques do tipo UDP Scan

Label	**Número de Fluxos**
Benign	33385
UDP Scan	19

Tabela 4.4: Cenário no qual a rede sofreu ataques tipo MQTT Brute-Force

Label	**Número de Fluxos**
Benign	2516
MQTT Brute-Force	2000264

Tabela 4.5: Cenário no qual a rede sofreu ataques do tipo Sparta SSH Brute-Force

Label	**Número de Fluxos**
Benign	149977
Sparta SSH Brute-Force	1013391

Por fim, antes do *dataset* estar pronto para ser processado pelos algoritmos, assim como em Pontes et al. [11] [13] são retiradas as *features Source IP*, *Destination IP*, *Flow ID* e *Timestamp*. *Source IP* e *Destination IP* são retirados pois foram inventadas no ambiente de simulação, enquanto *Flow ID* e *Timestamp* são retiradas por serem intrinsecamente não informativas. Feito isso, os arquivos CSV estão prontos para serem interpretados pelos algoritmos.

4.2 Algoritmos de Classificação

Nesta seção serão apresentados os algoritmos implementados no scikit-learn [33] 1.0.2 que servirão de comparação com o EFC, detalhando quais os parâmetros utilizados. Todos os quatro algoritmos a seguir demonstraram boa eficácia ao classificar fluxos de rede *IoT* [7] [6]. Além deles, também tentou-se utilizar os algoritmos *K-Nearest Neighbors* e SVM para classificar os fluxos do *dataset*, entretanto por limitações computacionais não foi possível testar nenhum dos dois. Utilizando o *K-Nearest Neighbors* sobre o *dataset* resultou em *segfault* por falta de memória. Já o SVM não é recomendado para *datasets* com mais de dezenas de milhares de amostras, devido ao fato do seu tempo de treinamento ser quadrático em relação ao número de amostras [34].

4.2.1 Gaussian Naive Bayes

O *Naive Bayes* é um classificador probabilístico, ou seja, dada uma amostra tem como retorno a distribuição de probabilidades desta amostra sobre o conjunto de classes do experimento, e não simplesmente a classe cuja probabilidade é maior [35]. Para tal, o algoritmo utiliza uma versão simplificada do Teorema de Bayes [36]. Este modelo assume forte independência entre as *features* da amostra, o que limita seus casos de uso. Entretanto, em Khraisat et al. [7] se mostrou eficaz para classificar fluxos de rede IoT.

Neste trabalho foi utilizada a implementação do *GaussianNB* com os parâmetros padrão da biblioteca. Para mais informações sobre os parâmetros, consultar a documentação da biblioteca em `https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html`

4.2.2 Decision Tree

O algoritmo *Decision Tree* [37] utiliza uma árvore de decisão para classificar o dado de entrada. Uma árvore de decisão é composta por nós de decisão e folhas. Nos nós de decisão estão os testes sobre as *features* das amostras e seus filhos são os resultados destes testes. As folhas representam as classes do experimento, que são o resultado final dado pelo algoritmo [38]. Existem vários critérios para decidir como as *features* devem ser distribuídas para construir a árvore, como por exemplo *Gini impurity* [39] e Entropia [40].

Neste trabalho foi utilizada a implementação do *DecisionTreeClassifier* com os parâmetros padrão da biblioteca, sendo o critério para a construção da árvore o *Gini impurity* mencionado no parágrafo anterior. Para mais informações sobre os parâmetros, consultar

a documentação da biblioteca em `https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html`

4.2.3 Random Forest

O algoritmo *Random Forest* é do tipo *ensemble*, ou seja, ele junta vários algoritmos preditivos para obter um melhor resultado. Ele combina várias árvores de decisão e devido ao fato de a construção de cada árvore ter certa aleatoriedade ele reduz o risco de *overfitting* [41]. O algoritmo possui ótimo desempenho para classificar fluxos de rede [42].

Neste trabalho foi utilizada a implementação do *RandomForestClassifier* com os parâmetros padrão da biblioteca, composto por cem árvores de decisão e sendo o critério para a construção da árvore o *Gini impurity*. Para mais informações sobre os parâmetros, consultar a documentação da biblioteca em `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`

4.2.4 Multilayer Perceptron

O algoritmo *Multilayer Perceptron* é um tipo de Artificial Neural Network (ANN). Uma ANN geralmente é composta de várias camadas de "neurônios" que ao receberem um valor de entrada, pesam a informação e calculam um novo valor para passar aos neurônios da próxima camada, que por sua vez repetem o processo [43]. As ANNs possuem regras de aprendizagem tal como *back-propagation* que permitem ao algoritmo propagar os erros inclusive para as camadas anteriores, o que possibilita o tratamento de relações complexas e não-lineares entre diferentes *features* da amostra [44].

Neste trabalho foi utilizada a implementação do *MLPClassifier* com os parâmetros padrão da biblioteca, tendo sua *hidden layer* composta de cem neurônios. Para mais informações sobre os parâmetros, consultar a documentação da biblioteca em `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`

4.2.5 Complexidade dos Algoritmos

Nessa Seção será analisada a complexidade de treinamento de todos os algoritmos utilizados neste trabalho. A complexidade de treinamento representa os fatores que influenciam no tempo de treinamento de um algoritmo. Saber esses fatores pode ajudar a escolher o algoritmo mais apropriado para uso em um certo contexto, assim como ajudar a melhorar esse tempo aplicando certas técnicas, como, por exemplo, pré-processamentos no *dataset* e diminuindo o número de *features*. A tabela se encontra no decorrer do texto:

Tabela 4.6: Tabela de complexidade de treinamento dos algoritmos

Algoritmos	Complexidade de Treinamento
EFC	$\mathcal{O}((M \times Q)^3 + N \times M^2 \times Q^2)$ [11]
Gaussian Naive Bayes	$\mathcal{O}(M \times N)$ [14]
Decision Tree	$\mathcal{O}(M \times N^2)$ [14]
Random Forest	$\mathcal{O}(X \times M \times N \times logN)$ [14]
Multilayer Perceptron	$\mathcal{O}(E \times M \times N \times K)$ [14]

Na Tabela 4.6, M é o número de *features* das amostras, N é a quantidade de amostras, Q é o tamanho do alfabeto do EFC, X é a quantidade de árvores do *Random Forest* e E e K são a quantidade de épocas e neurônio respectivamente do *Multilayer Perceptron*. Observando o tempo de treinamento do EFC, é possível ver que ele é baixo, linear sobre a quantidade N de amostras, menor se comparado ao *Decision Tree*, *Multilayer Perceptron* e *Random Forest* mas maior se comparado ao *Gaussian Naive Bayes*. A complexidade do EFC é considerada dominada pelo termo $N \times M^2 \times Q^2$ uma vez que se espera que o número de amostras seja muito maior do que a quantidade de *features* e do tamanho do alfabeto, o que implica que $(M \times Q)^3$ geralmente não é dominante sobre $N \times M^2 \times Q^2$. Isso faz com que a complexidade seja mais dependente da quantidade de *features* escolhidas, ressaltando a utilidade de métodos como *feature selection* para melhorar seu desempenho, embora isso não seja tratado neste trabalho.

4.3 Desenho dos Experimentos

Nesta seção serão descritos os experimentos deste trabalho, que podem ser divididos em três diferentes tipos. No primeiro, foi testada a eficácia do EFC Binário, executando-o em cada um dos cenários de ataque fornecidos no *dataset* com o intuito de ver se ele é efetivo em classificar estes tipos de fluxo. No segundo, foi testada a eficácia do EFC multi-classe executando-o sobre um arquivo CSV contendo os fluxos de todos os cenários ao mesmo tempo e comparando com os algoritmos apresentados na Seção 4.2. Já o terceiro foi testado de maneira análoga ao segundo, com a diferença que uma classe de ataque foi retirada do treinamento para poder simular fluxos de ataque desconhecidos na etapa de teste.

Nenhuma das implementações dos algoritmos utilizados neste trabalho conseguem lidar muito bem com *strings* [45]. Tanto as implementações do *scikit-learn* quanto a do EFC conseguem lidar somente com dados numéricos. Entretanto, nos arquivos CSV criados a partir do *dataset*, a coluna de *labels* possui valores em *string*. Portanto, é necessário converter estes dados categóricos correlacionando cada *string* diferente a um valor numérico único. Para isso, foi utilizada a função *LabelEncoder* [46] presente no

próprio *scikit-learn*, que além de ser rápida e fácil de implementar, permite reverter a conversão ao final da classificação, facilitando o entendimento dos resultados ao olho humano.

4.3.1 Classificação Binária

Como explicado na Seção 2.1.1, o EFC Multi-classe é a combinação de vários EFC Binários, um para cada classe do experimento [13]. Isso faz com que saber a eficácia do EFC Binário em classificar corretamente fluxos benignos e maliciosos seja relevante para este trabalho. Portanto, o algoritmo foi aplicado em todos os cenários de ataque do *dataset* com exceção do cenário no qual a rede sofreu ataques do tipo *UDP Scan*. Esta decisão foi tomada pois, como é possível observar na Tabela 4.3, a relação de fluxos benignos para fluxos maliciosos deste cenário é muito desbalanceada, o que impactaria na eficácia dos algoritmos e tornaria qualquer gráfico gerado muito difícil de visualizar. Para cada cenário de ataque foi considerado 70% dos fluxos para treinamento e 30% para teste.

Na primeira etapa é inferido o modelo de classificação a partir do conjunto de amostras de treinamento. Depois, tendo como entrada os fluxos de teste, o algoritmo retorna a predição das classes desses fluxos e os valores da energia de cada fluxo [11]. Mas somente com estes valores não é possível chegar a nenhuma conclusão. Portanto, foi utilizada a função *classification_report* [47] do *scikit-learn* que compara os valores esperados das classes de teste com os valores preditos pelo algoritmo e retorna três métricas: *precision*, *recall* e *f1-score* [48]. Estas são as métricas que servirão como base para analisar o desempenho de todos os algoritmos utilizados neste trabalho, e não somente o desempenho do EFC.

Antes de mostras como essas métricas são calculadas, é necessário explicar o conceito de verdadeiros positivos, verdadeiros negativos, falsos positivos e falsos negativos. Observando a matriz de confusão da Figura 2.6, o verdadeiro positivo na parte superior-esquerda representa a quantidade de fluxos benignos que foram corretamente classificados. O verdeiro negativo na parte inferior-direita representa a quantidade de fluxos maliciosos que foram corretamente classificados. Consequentemente, as denominações falso positivo e falsos negativo são a quantidade de fluxos benignos e maliciosos classificados erroneamente, respectivamente. Nesse tipo de matriz de confusão, quanto mais escura for a matriz identidade, melhor o desempenho do algoritmo, podendo facilmente ser expandida para um contexto multi-classe.

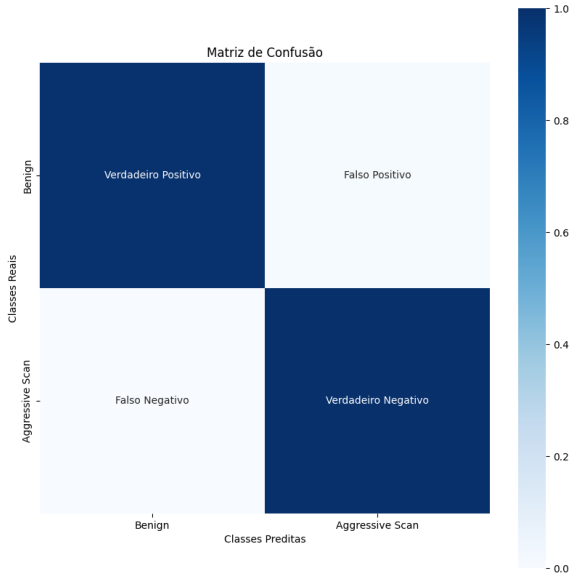

Figura 4.1: Matriz de confusão.

Com isso já é possível explicar as métricas utilizadas. A *precision* pode ser calculada utilizando a seguinte equação:

$$precision = \frac{tp}{(tp + fp)} \tag{4.1}$$

onde tp é o número de verdadeiros positivos e fp é o número de falsos positivos. Em outras palavras, a *precision* é utilizada para avaliar o desempenho do algoritmo em classificar corretamente os fluxos maliciosos [47]. Os valores vão de 0 a 1, sendo 0.00 o pior e 1.00 o melhor. Já o *recall* é possível calcular utilizando a seguinte equação:

$$recall = \frac{tp}{(tp + fn)} \tag{4.2}$$

onde tp é o número de verdadeiros positivos e fn é o número de falsos negativos. Em outras palavras, o *recall* é utilizado para avaliar o desempenho do algoritmo em classificar corretamente os fluxos benignos [47]. Os valores vão de 0 a 1, sendo 0.00 o pior e 1.00 o melhor. E por fim, o *f1-score* é a média harmônica das métricas *precision* e *recall*, dado por:

$$f1 = \frac{2 * (precision * recall)}{precision * recall} \tag{4.3}$$

Como podemos observar na Seção 4.1, muitos dos cenários possuem uma relação entre quantidade de fluxos benignos e fluxos maliciosos completamente desbalanceada, o que pode influenciar bastante os resultados de algumas métricas. Para tanto, é proposta a média ponderada dos resultados como uma nova métrica, pois leva em conta todos os fluxos do cenário e não de somente uma única classe [48]. Isso faz que com a flutuação de desempenho de uma única classe não afete o desempenho geral se a quantidade de fluxos classificados erroneamente não for relevante para o todo do cenário. Ela é dada por:

$$\frac{\sum_{i=1}^{n} class_metric_i * class_samples_i}{total_samples} \tag{4.4}$$

onde $class_metric_i$ é o valor da métrica da classe pertencente ao conjunto i de classes do cenário, $class_samples_i$ é a quantidade de amostras da classe i e $total_samples$ é a quantidade total de amostras do cenário.

Para ilustrar este processo de classificação de modo visual, são plotados gráficos utilizando os valores da energia de cada fluxo e o valor de *cutoff* das energias calculado pelo algoritmo. Nestes gráficos, todos os valores à esquerda da linha de corte são considerados como benignos e todos os valores à direita considerados como maliciosos.

Neste trabalho foi utilizada a biblioteca *EnergyBasedFlowClassifier*, disponibilizada em um repositório público do *GitHub* [49], sendo utilizado primeiramente o valor do *cutoff_quantile* padrão da biblioteca de 0.95 e repetido o experimento com outros valores para decidir qual utilizar nos experimentos multi-classe. O parâmetro *base_class* é 1 para o cenário no qual a rede sofreu ataques do tipo *Aggressive Scan* e 0 para o restante dos cenários. O restante dos parâmetros são padrão da biblioteca. Para mais informações sobre os parâmetros, consultar a documentação da biblioteca em `https://efc-package.readthedocs.io/en/latest/generated/efc.EnergyBasedFlowClassifier.htm`

4.3.2 Classificação Multi-classe

A grande diferença de um classificador multi-classe para um binário é que dada uma amostra com mais de duas classes, o multi-classe é capaz de identificar cada tipo de classe em que foi treinado enquanto o binário consegue reconhecer somente a classe única com que foi treinado, descartando todo o resto como malicioso [50]. Entretanto, todos os cenários presentes no *dataset* possuem somente duas classes, o que não propicia o teste de classificadores multi-classe. Para remediar isso, utilizando a biblioteca *pandas* [51] do Python, foram combinados todos os cenários do *dataset* em um único arquivo CSV, com exceção do cenário no qual a rede sofreu ataques do tipo *UDP Scan* por causa do desbalanceamento da classe. Este arquivo único possui, portanto, as classes *Aggressive Scan*, *Benign*, *MQTT Brute-Force* e *Sparta SSH Brute-Force*.

Nesta etapa do processo não se quer somente medir o desempenho isolado do EFC, mas também compará-lo a outros classificadores. Para tanto, são utilizados os algoritmos citados na Seção 4.2 e as mesma métricas utilizadas para o EFC Binário citadas na Seção 4.3.1. Entretanto, nos testes com o EFC multi-classe não serão plotados os gráficos de energia, uma vez que não é possível criar gráficos comparáveis para os outros algoritmos.

Neste trabalho foi utilizada a biblioteca *EnergyBasedFlowClassifier*, disponibilizada em um repositório público do *GitHub* [49], com o valor do *cutoff_quantile* sendo o melhor valor encontrado no experimento anterior. O restante dos parâmetros são padrão da biblioteca. Para mais informações sobre os parâmetros, consultar a documentação da biblioteca em `https://efc-package.readthedocs.io/en/latest/generated/efc.EnergyBasedFlowClassifier.htm`

4.3.3 Classificação Multi-classe com ataques desconhecidos

Uma das características que diferencia o EFC dos outros algoritmos é que ele é capaz de reconhecer ataques desconhecidos [13]. Entretanto, como simular um ataque desconhecido? A resposta encontrada foi retirar um dos tipos de ataque do conjunto de treinamento, mas deixá-lo no conjunto de teste. Desta forma, como o algoritmo nunca foi treinado com este ataque, na etapa de teste ele age como um ataque desconhecido. O processo é repetido três vezes, uma vez para cada tipo de ataque presente no *dataset*.

Com exceção da retirada das classes do conjunto de treinamento, o processo de teste é essencialmente o mesmo descrito na Seção 4.3.2. Os algoritmos e as métricas utilizados também são os mesmos. Entretanto, os outros algoritmos que serão comparados ao EFC não possuem outra maneira de lidar com ataques desconhecidos que não seja classificá-los como uma das quatro classes com que foram treinados. Deste modo, a função *classification_report* considerará um caso em que o ataque desconhecido for classificado como qualquer uma das classes de ataque restantes como um falso positivo, o que é injusto. Portanto, os valores de *precision*, *recall* e *f1-score* serão recalculados contando como falsos positivos somente os fluxos de ataque desconhecidos que foram classificados como fluxos benignos. E por fim, para ilustrar este processo de classificação de modo visual, serão plotados gráficos para cada classe, mostrando os fluxos de cada classe presente no *dataset* e em que classes o algoritmo de fato os classificou.

Capítulo 5

Resultados

Neste capítulo serão apresentados os resultados dos experimentos utilizando o EFC e os classificadores de ML detalhados na Seção 4.2. Começa-se analisando os resultados da classificação binária utilizando somente o EFC para provar que o algoritmo é capaz de separar fluxos benignos e maliciosos do *dataset* utilizado. Em seguida, serão analisados os testes comparativos da versão multi-classe do EFC com os outros algoritmos de ML. E, por último, também serão analisados os resultados comparativos do EFC multi-classe com os outros algoritmos de ML, mas com uma das classes do *dataset* removidas da etapa de treino dos algoritmos, de modo a simular um ataque desconhecido.

5.1 Classificação Binária

Como explicado na Seção 4.3.1, o algoritmo do EFC Binário foi aplicado a cada cenário de ataque do *dataset*, com exceção do cenário no qual a rede sofreu ataques do tipo *UDP Scan*. Iniciou-se a classificação utilizando os parâmetros padrão da biblioteca do EFC, o que inclui um *cutoff_quantile* de 0.95. A partir dos resultados, foram calculadas as métricas *precision*, *recall* e *f1-score* e plotados gráficos para ajudar a entender o impacto que o *cutoff_quantile* tem nos resultados. As tabelas e os gráficos deste experimento estão na sequência do texto:

5.1.1 Utilizando *cutoff_quantile* de 0.95

Tabela 5.1: Métricas de desempenho do EFC no cenário de ataque Aggressive Scan

Label	**Precision**	**Recall**	**F1-Score**	**Nº de Amostras**
Aggressive Scan	0.97	1.00	0.98	6018
Benign	1.00	0.94	0.97	3356
Média Ponderada	0.98	0.98	0.98	9374

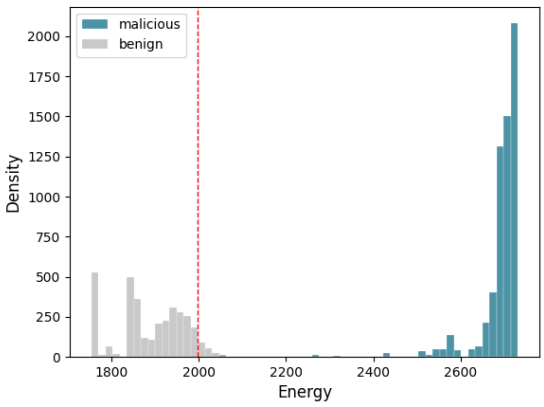

Figura 5.1: Gráfico das energias do EFC no cenário de ataque Aggressive Scan

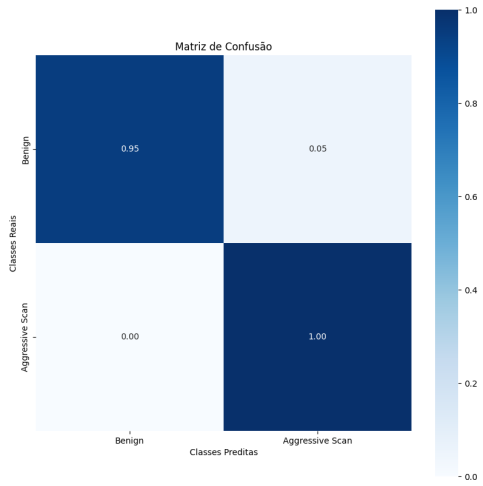

Figura 5.2: Matriz de confusão do EFC no cenário de ataque Aggressive Scan

Analisando os dados da Tabela 5.1, é possível perceber que o EFC teve um ótimo desempenho no cenário no qual a rede sofreu ataques do tipo *Aggressive Scan*. A classe *Benign* tendo *precision* de 1.00 e a classe *Aggressive Scan* tendo *recall* de 1.00 indica que, de modo geral, todos os fluxos maliciosos foram classificados corretamente. Embora o mesmo não seja verdade para os fluxos benignos, a taxa de falsos negativos está dentro de um limite aceitável.

Já analisando o gráfico da Figura 5.1, é possível ver uma clara separação entre as energias de fluxos benignos e maliciosos. Entretanto, a linha de corte que representa o *cutoff* das energias está incidindo em um valor anterior ao das últimas energias que representam fluxos benignos. Isso indica que caso o valor de *cutoff* fosse maior, a quantidade de falsos negativos provavelmente seria menor.

Tabela 5.2: Métricas de desempenho do EFC no cenário de ataque MQTT Brute-Force

Label	**Precision**	**Recall**	**F1-Score**	**Nº de Amostras**
Benign	0.78	0.92	0.85	755
MQTT Brute-Force	1.00	1.00	1.00	600079
Média Ponderada	1.00	1.00	1.00	600834

Em relação ao cenário no qual a rede sofreu ataques do tipo *MQTT Brute-Force*, observando as métricas da Tabela 5.2, poderia se pensar que o desempenho do EFC caiu bastante se comparado ao cenário anterior. A *precision* de 0.78 da classe *Benign* indica que vários fluxos maliciosos foram erroneamente classificados como fluxos benignos, o que em um contexto de classificadores para NIDSs é um problema. Entretanto, ao se analisar

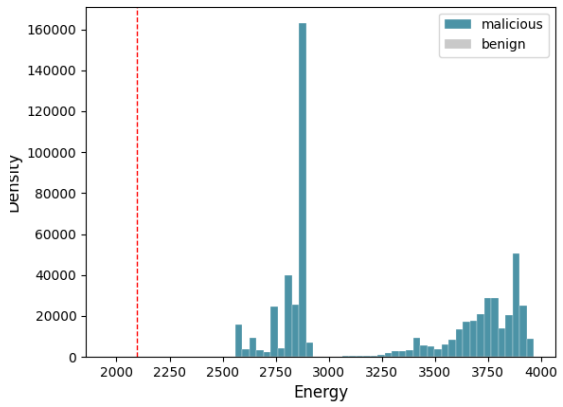

Figura 5.3: Gráfico das energias do EFC no cenário de ataque MQTT Brute-Force

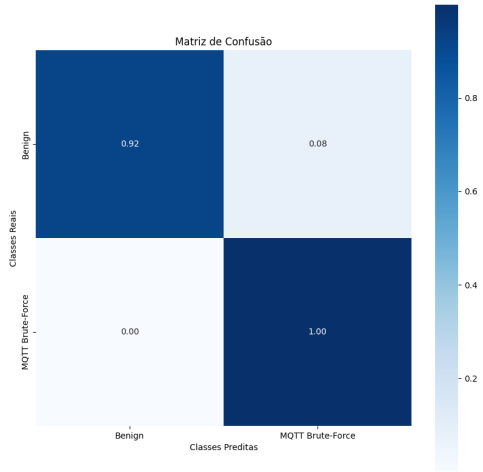

Figura 5.4: Matriz de confusão do EFC no cenário de ataque MQTT Brute-Force

o valor de 1.00 do *recall* da classe *MQTT Brute-Force*, existe um conflito, pois este número indica que quase todos os fluxos maliciosos foram corretamente classificados. Isso ocorre por que a distribuição de fluxos está muito desbalanceada a favor dos fluxos maliciosos, então mesmo que uma porcentagem ínfima dos fluxos maliciosos seja erroneamente classificada como benigna, já é o suficiente para impactar bastante as métricas dos fluxos benignos.

Para resolver isso, será levada em conta a média ponderada, explicada na Seção 4.3.1, que ao aproximar todas as métricas para 1.00, indica que de modo geral o EFC realizou um bom trabalho classificando os fluxos. O problema do desbalanceamento da quantidade de fluxos também afeta de forma negativa o gráfico da Figura 5.3, uma vez que os fluxos benignos praticamente não estão visíveis. Apesar disto, percebe-se que os fluxos maliciosos estão claramente à direita da linha de corte, o que confirma o que foi visto nas métricas de que praticamente todos os fluxos maliciosos foram corretamente classificados.

Tabela 5.3: Métricas de desempenho do EFC no cenário de ataque Sparta SSH Brute-Force

Label	**Precision**	**Recall**	**F1-Score**	**Nº de Amostras**
Benign	1.00	0.95	0.98	44993
Sparta SSH Brute-Force	0.99	1.00	1.00	304012
Média Ponderada	0.99	0.99	0.99	349005

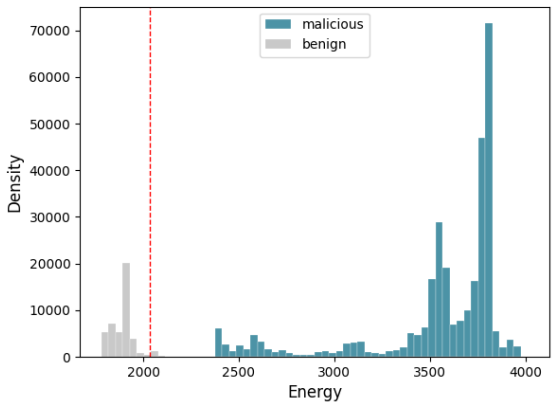

Figura 5.5: Gráfico das energias do EFC no cenário de ataque Sparta SSH Brute-Force

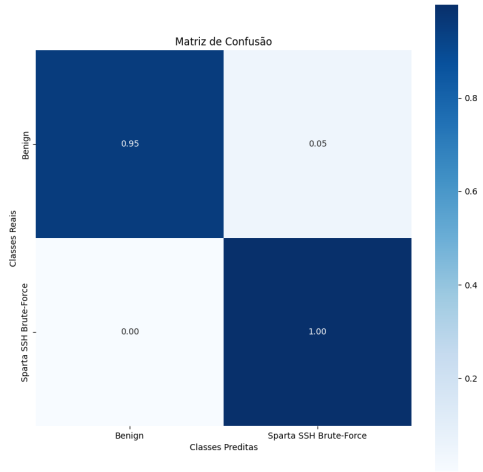

Figura 5.6: Matriz de confusão do EFC no cenário de ataque Sparta SSH Brute-Force

Por fim, serão analisados os resultados da tabela Tabela 5.3, que trata do cenário no qual a rede sofreu ataques do tipo *Sparta SSH Brute-Force*. De forma análoga ao cenário no qual a rede sofreu ataques do tipo *Aggressive Scan*, os fluxos benignos possuem um *precision* de 1.00 e os fluxos maliciosos um *recall* de também 1.00, o que indica um ótimo desempenho. Os falsos negativos também estão em uma taxa aceitável.

O gráfico da Figura 5.5 também segue a mesma análise do gráfico da figura Figura 5.1, no qual há uma clara separação entre as energias dos fluxos benignos e dos maliciosos, o que dá margem para aumentar o valor do *cutoff* para possivelmente diminuir a quantidade de falsos negativos. Portanto, na próxima seção, serão repetidos os experimentos, mas com o valor do *cutoff_quantile* sendo 0.99. As tabelas e os gráficos deste experimento estão na sequência do texto:

5.1.2 Utilizando *cutoff_quantile* de 0.99

Tabela 5.4: Métricas de desempenho do EFC no cenário de ataque Aggressive Scan

Label	Precision	Recall	F1-Score	Nº de Amostras
Aggressive Scan	0.99	1.00	0.98	6018
Benign	1.00	0.99	0.99	3356
Média Ponderada	1.00	1.00	1.00	9374

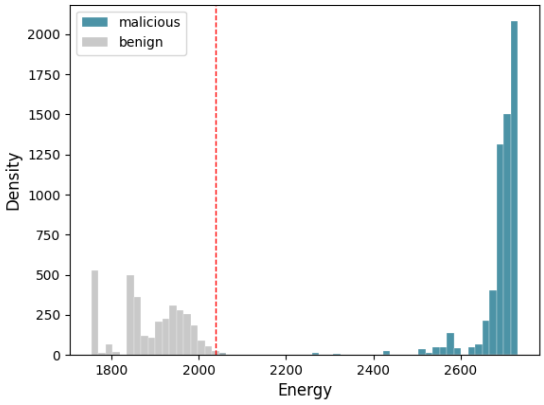

Figura 5.7: Gráfico das energias do EFC no cenário de ataque Aggressive Scan

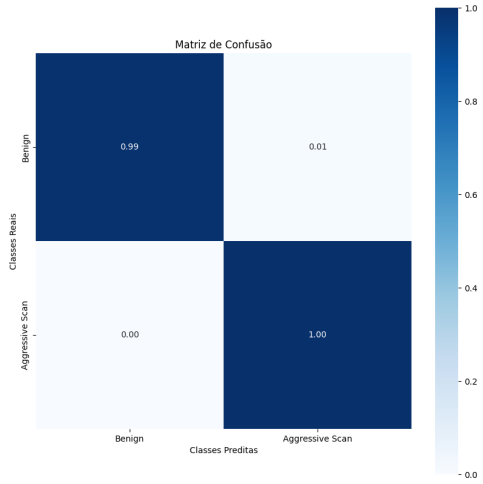

Figura 5.8: Matriz de confusão do EFC no cenário de ataque Aggressive Scan

Comparando a Tabela 5.4 com a Tabela 5.1, é possível perceber uma melhora de desempenho do EFC no cenário no qual a rede sofreu ataques do tipo *Aggressive Scan*, com a média ponderada de todas as métricas passando de 0.98 para 1.00. Isso tem a ver com a diminuição de falsos positivos, como é possível ver pelo aumento do *recall* da classe *Benign* de 0.94 para 0.99. Pode-se confirmar isso também ao observar o gráfico da Figura 5.7, que se comparado ao da Figura 5.1 quase não possui fluxos à direita do *cutoff*.

Tabela 5.5: Métricas de desempenho do EFC no cenário de ataque MQTT Brute-Force

Label	**Precision**	**Recall**	**F1-Score**	**Nº de Amostras**
Benign	0.79	0.98	0.87	755
MQTT Brute-Force	1.00	1.00	1.00	600079
Média Ponderada	1.00	1.00	1.00	600834

Ao comparar a Tabela 5.5 com a Tabela 5.2, é possível perceber uma melhora de desempenho do EFC. Embora a média ponderada continue a mesma, o *recall* da classe *Benign* aumentou de 0.92 para 0.98, denotando diminuição na taxa de falsos negativos. Já a *precision* da classe *Benign*, que denota a quantidade de falso positivos, continua praticamente a mesma, o que, da mesma forma que explicado na Seção 5.1.1, tem a ver com a distribuição de fluxos estar muito desbalanceada a favor dos fluxos maliciosos. Este problema também afeta o gráfico da Figura 5.9, no qual quase não é possível ver os fluxos benignos, dificultando qualquer análise visual das mudanças indicadas pelas métricas da Tabela 5.5.

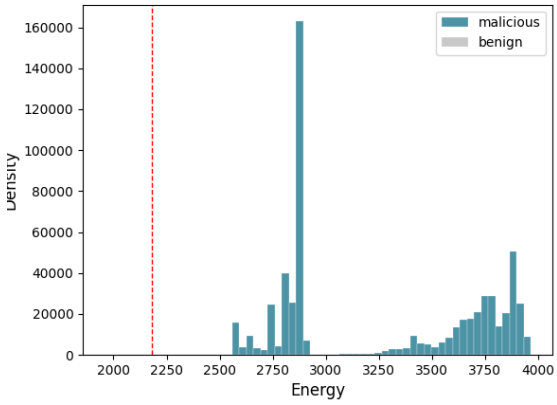

Figura 5.9: Gráfico das energias do EFC no cenário de ataque MQTT Brute-Force

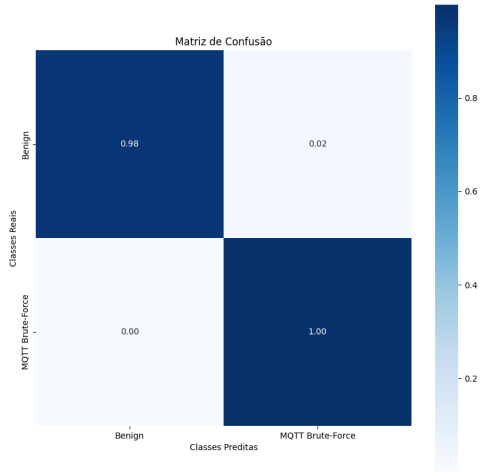

Figura 5.10: Matriz de confusão do EFC no cenário de ataque MQTT Brute-Force

Tabela 5.6: Métricas de desempenho do EFC no cenário de ataque Sparta SSH Brute-Force

Label	**Precision**	**Recall**	**F1-Score**	**Nº de Amostras**
Benign	1.00	0.99	0.99	44993
Sparta SSH Brute-Force	1.00	1.00	1.00	304012
Média Ponderada	1.00	1.00	1.00	349005

A análise do cenário no qual a rede sofreu ataques do tipo *Sparta SSH Brute-Force* é muito parecida com análise feita para os ataques *Aggressive Scan* nesta seção. Ao comparar a Tabela 5.6 com a Tabela 5.3, percebe-se melhora no desempenho do EFC, com a média ponderada de todas as métricas passando de 0.99 para 1.00. Isso tem a ver com a diminuição de falsos negativos, refletido no aumento do *recall* da classe *Benign* de 0.95 para 0.99. Pode-se confirmar isso também ao observar o gráfico da Figura 5.6, que se comparado ao da Figura 5.3 quase não possui fluxos à direita do *cutoff*.

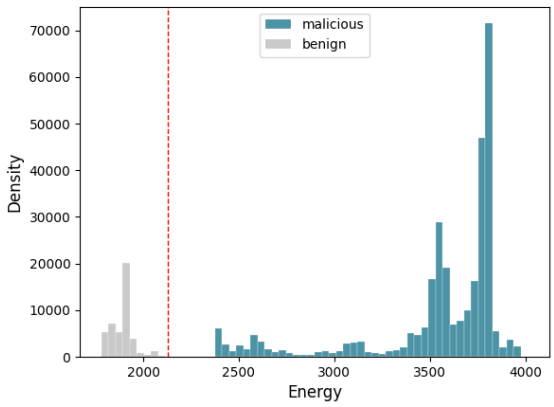

Figura 5.11: Gráfico das energias do EFC no cenário de ataque Sparta SSH Brute-Force

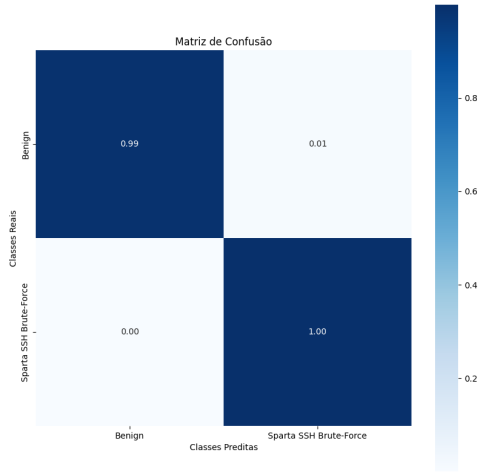

Figura 5.12: Matriz de confusão do EFC no cenário de ataque Sparta SSH Brute-Force

Como os resultados com o *cutoff_quantile* do algoritmo EFC sendo 0.99 são objetivamente melhores do que os resultados com o valor padrão de 0.95, para os experimentos multi-classe a seguir será utilizado o valor de 0.99.

5.2 Classificação Multi-classe

Como explicado na Seção 4.3.2, foi necessário combinar todos os cenários em um único arquivo CSV para poder executar tanto o EFC Multi-classe quanto os outros algoritmos de comparação. Com exceção do *cutoff_quantile* de 0.99, foram utilizados os parâmetros padrão da biblioteca do EFC para este experimento. A partir dos resultados foram calculadas as métricas *precision*, *recall* e *f1-score*. As tabelas deste experimento estão a seguir:

Tabela 5.7: Métricas de desempenho do *Decision Tree*

Decision Tree				
Labels	**Precision**	**Recall**	**F1-Score**	**Nº de Amostras**
Aggressive Scan	1.00	1.00	1.00	6017
Benign	1.00	1.00	1.00	99252
MQTT Brute-Force	1.00	1.00	1.00	600080
Sparta SSH Brute-Force	1.00	1.00	1.00	304012
Média Ponderada	1.00	1.00	1.00	1009361

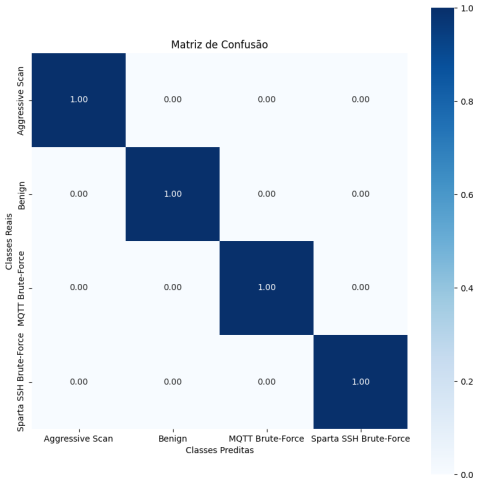

Figura 5.13: Matriz de confusão do *Decision Tree*.

Os resultados do *Decision Tree*, apresentados na Tabela 5.7, surpreendem. Com todas as métricas com valor 1.00, o algoritmo foi praticamente perfeito em classificar todos os fluxos de rede do *dataset*. Embora treinamento e teste dentro do mesmo domínio com todas as classes conhecidas não seja o teste definitivo para saber se um algoritmo é ideal para uso em um NIDSs, este resultado é encorajador para os próximos testes, no qual é preciso saber se ele manterá a mesma taxa de acerto.

Tabela 5.8: Métricas de desempenho do EFC Multi-classe

EFC Multi-classe				
Labels	**Precision**	**Recall**	**F1-Score**	**Nº de Amostras**
Aggressive Scan	0.98	1.00	0.99	6017
Benign	0.94	0.99	0.96	99252
MQTT Brute-Force	1.00	0.99	0.99	600080
Sparta SSH Brute-Force	1.00	1.00	1.00	304012
Média Ponderada	0.99	0.99	0.99	1009361

Observado os resultados do EFC Multi-classe na Tabela 5.8, a maioria das métricas é muito boa, mas com destaque negativo para a *precision* da classe *Benign*, que assume um valor de 0.94 abaixo da média das outras métricas. Analisando melhor a tabela, pode-se deduzir que ela é mais baixa porque uma pequena parcela dos fluxos *MQTT Brute-Force* foi classificada erroneamente, visto o *recall* da classe de 0.99. Entretanto, como a quantidade de amostras de fluxos *MQTT Brute-Force* é bem maior que a de fluxos benignos, qualquer parcela de fluxos classificados erroneamente como benignos tem

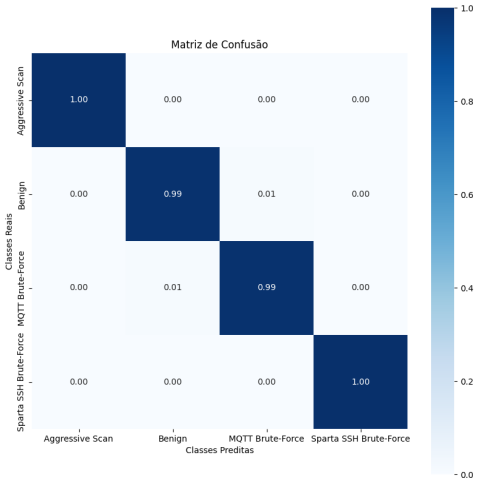

Figura 5.14: Matriz de confusão do EFC.

a capacidade de mudar significativamente a métrica *precision* da classe *Benign*. Quando isso ocorre, é recomendável utilizar a média ponderada das métricas, assim como foi feito na análise da Tabela 5.2. Com a média ponderada de todas métricas sendo 0.99, é possível afirmar que o EFC obteve um desempenho excelente em todas as classes.

Tabela 5.9: Métricas de desempenho do *Gaussian Naive Bayes*

Gaussian Naive Bayes				
Labels	**Precision**	**Recall**	**F1-Score**	**Nº de Amostras**
Aggressive Scan	0.02	1.00	0.03	6017
Benign	0.00	0.00	0.00	99252
MQTT Brute-Force	0.94	0.36	0.52	600080
Sparta SSH Brute-Force	0.66	0.87	0.75	304012
Média Ponderada	0.76	0.48	0.54	1009361

Já os resultados do *Gaussian Naive Bayes* na Tabela 5.9 surpreendem negativamente, tendo em vista o valor de 0.00 em todas as três métricas de desempenho ao tentar classificar os fluxos benignos. Isso significa que essencialmente qualquer fluxo benigno detectado pelo *Gaussian Naive Bayes* seria classificado como algum tipo de fluxo malicioso, o que em um contexto para uso em um NIDSs é um dos piores cenários possíveis.

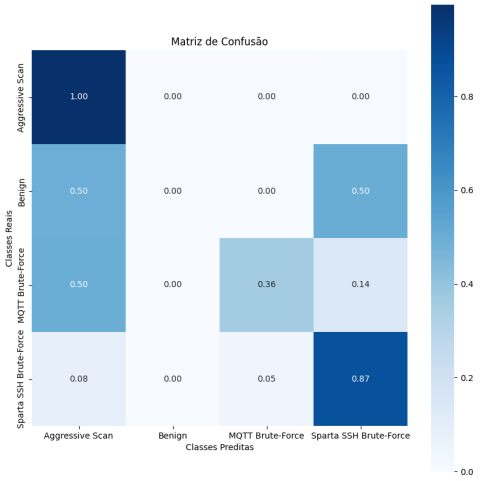

Figura 5.15: Matriz de confusão do *Gaussian Naive Bayes*.

Tabela 5.10: Métricas de desempenho do *Multilayer Perceptron*

Multilayer Perceptron				
Labels	**Precision**	**Recall**	**F1-Score**	**Nº de Amostras**
Aggressive Scan	1.00	0.99	1.00	6017
Benign	1.00	0.50	0.66	99252
MQTT Brute-Force	0.65	1.00	0.78	600080
Sparta SSH Brute-Force	1.00	0.08	0.15	304012
Média Ponderada	0.79	0.67	0.58	1009361

Os resultados do *Multilayer Perceptron* na Tabela 5.10 mostram que o algoritmo teve certa dificuldade. O valor de 0.65 da *precision* da classe *MQTT Brute-Force* e o valor de 0.08 do *recall* da classe *Sparta SSH Brute-Force* indicam que grande parte dos fluxos *Sparta SSH Brute-Force* foram classificados erroneamente como *MQTT Brute-Force*. Já na classe *Benign*, embora ele faça um excelente trabalho evitando falsos positivos, como denota a *precision* de 1.00, ele sofre bastante com falsos negativos, evidenciado pelo *recall* de 0.50. Isso significa que metade dos fluxos benignos analisados pelo algoritmo seriam classificados como fluxos maliciosos, o que é longe do ideal para aplicação em uma NIDS.

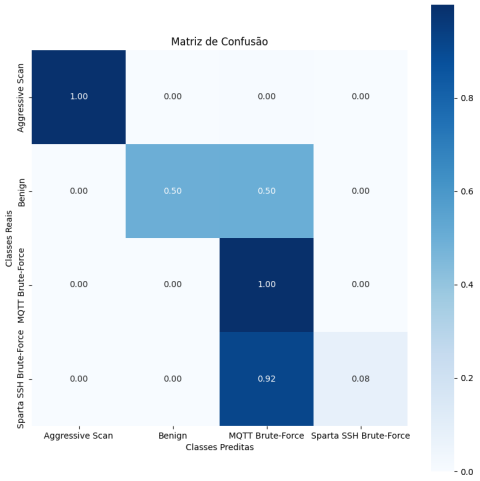

Figura 5.16: Matriz de confusão do *Multilayer Perceptron*.

Tabela 5.11: Métricas de desempenho do *Random Forest*

Random Forest				
Labels	**Precision**	**Recall**	**F1-Score**	**Nº de Amostras**
Aggressive Scan	1.00	1.00	1.00	6017
Benign	1.00	1.00	1.00	99252
MQTT Brute-Force	1.00	1.00	1.00	600080
Sparta SSH Brute-Force	1.00	1.00	1.00	304012
Média Ponderada	1.00	1.00	1.00	1009361

Já os resultados do *Random Forest* presentes na Tabela 5.11 são muito semelhantes ao apresentados pelo *Decision Tree*. Com todas as métricas com valor 1.00, ambos os algoritmos foram praticamente perfeitos em classificar todos os fluxos de rede do *dataset*. Da mesma forma, embora treinamento e teste dentro do mesmo domínio com todas as classes conhecidas não seja o teste definitivo para saber se um algoritmo é ideal para uso em um NIDSs, este resultado é encorajador para os próximos testes, nos quais é preciso saber se ele manterá essa taxa de acerto.

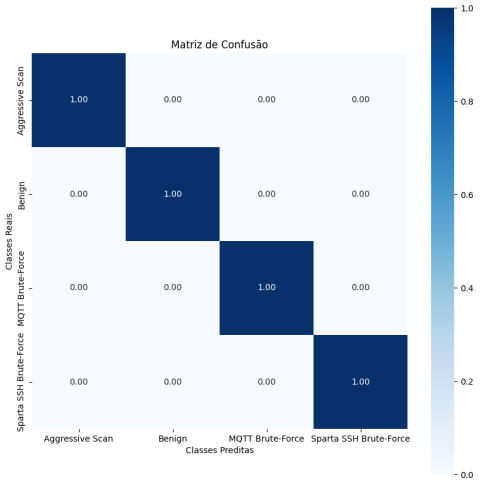

Figura 5.17: Matriz de confusão do *Random Forest*.

5.3 Classificação Multi-classe com ataques desconhecidos

Como explicado na Seção 4.3.3, para testar o desempenho dos algoritmos em cenários nos quais existe um ataque desconhecido é utilizado o arquivo CSV gerado para o teste do EFC multi-classe, no qual são retiradas todas as amostras de uma classe no conjunto de treinamento mas mantidas no conjunto de teste. Isto é repetido para cada uma das três classes de ataque presentes no *dataset*. Entretanto, para não ser injusto com os algoritmos de comparação, para eles só serão considerados como falsos positivos os fluxos desconhecidos que forem classificados como fluxos benignos, uma vez que eles não possuem uma classe própria para fluxos desconhecidos como o EFC. Foram também criados gráficos mostrando onde os fluxos de cada classe foram classificado, um para cada algoritmo.

5.3.1 Fluxos *Aggressive Scan* como ataques desconhecidos

Nesta etapa do experimento foram retirados os fluxos da classe *Aggressive Scan* do conjunto de treinamento para que servissem como fluxos desconhecidos, e foram criados gráficos e tabelas para que se possa analisar o desempenho de cada algoritmo. Entretanto, antes dos gráficos serem analisados, será feita uma breve explicação de como eles estão organizados. As barras do gráfico representam, em porcentagem, os fluxos de ataque presentes no conjunto de teste, enquanto as cores que preenchem as barras representam em que classe o algoritmo as classificou. A tabela abaixo das barras do gráfico explicita

a quantidade de cada tipo de fluxo que o algoritmo classificou em cada uma das classes representadas pelas barras.

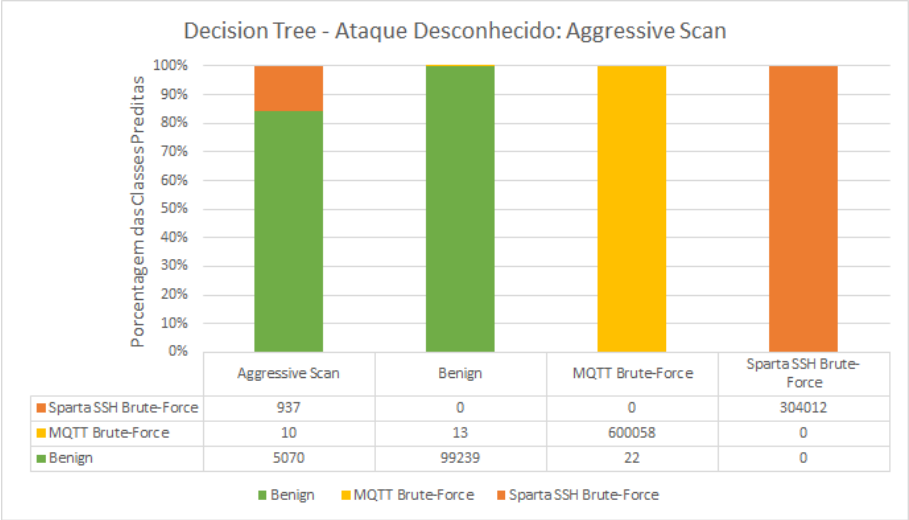

Figura 5.18: Gráfico de distribuição de classes do *Decision Tree*.

Tabela 5.12: Métricas de desempenho do *Decision Tree*

Labels	Precision	Recall	F1-Score
Benign	0.95	0.99	0.98
MQTT Brute-Force	0.99	0.99	0.99
Sparta SSH Brute-Force	1.00	1.00	1.00
Média Ponderada	0.99	0.99	0.99

Observando as colunas do gráfico da Figura 5.18 é possível notar que houve um ótimo desempenho na classificação dos fluxos cuja classe era conhecida, o que é confirmado pela média ponderada dos *recalls* de 0.99, presente na Tabela 5.12. Entretanto, observando novamente o gráfico, é possível ver que a maioria dos fluxos de *Aggressive Scan* foram classificados como fluxos benignos, o que é refletido pela *precision* de 0.95 da classe *Benign*. Embora este número não seja ruim, no contexto de uso para NIDSs, classificar um ataque desconhecido como um fluxo benigno não é ideal.

Tabela 5.13: Métricas de desempenho do EFC

Labels	Precision	Recall	F1-Score
Unknown	0.40	0.99	0.57
Benign	0.96	0.98	0.97
MQTT Brute-Force	0.99	0.98	0.99
Sparta SSH Brute-Force	0.99	0.99	0.99
Média Ponderada	0.99	0.99	0.99

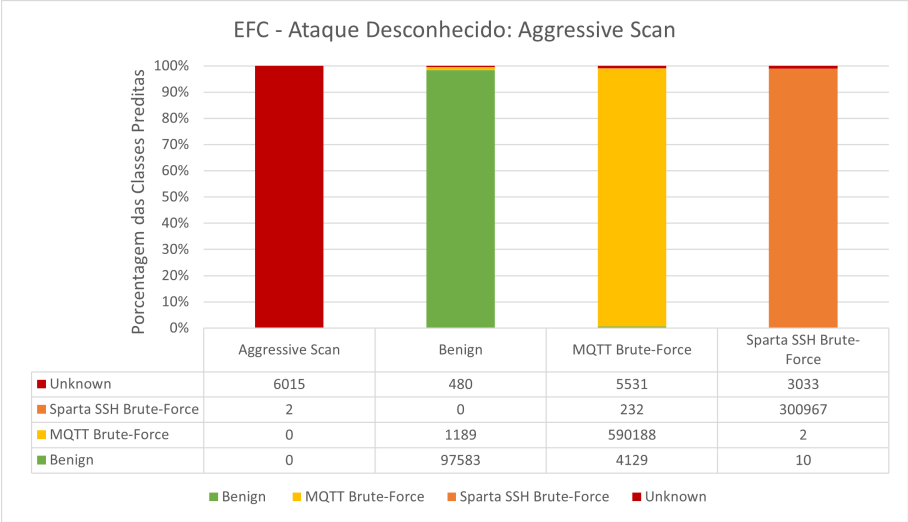

Figura 5.19: Gráfico de distribuição de classes do EFC.

Enquanto o restante dos gráficos e das tabelas desta seção possuem basicamente o mesmo layout, os do *EFC* são um pouco diferentes devido à presença da classe *Unknown*. Como explicado na Seção 2.1.1, se durante a classificação de um fluxo, na classe em que retornar a menor energia, a energia for maior que o *cutoff*, ele é classificado como suspeito, o que gera uma nova classe *Unknown*. Isso se traduz como uma nova cor nas barras do gráfico e uma nova linha na tabela de fluxos.

Observando as colunas do gráfico da Figura 5.19, é possível perceber que o EFC realizou um ótimo trabalho de classificação. A grande maioria dos fluxos conhecidos estão corretamente classificados e os fluxos de ataque desconhecido foram classificado quase em sua totalidade como *Unknown*, que é o resultado ótimo. Isto é corroborado pelo fato de que nenhum *recall* presente na Tabela 5.13 é menor do que 0.98.

Salta aos olhos o valor baixo de 0.40 da *precision* da classe *Unknown*. Isto ocorre devido ao fato de a classe desconhecida possuir poucas amostras, assim a pequena porcentagem de fluxos de outras classes que são classificados como *Unknown* impactam consideravelmente esta métrica. Entretanto, se for analisada a média ponderada da *precision* na Tabela 5.13, que leva em consideração todos os fluxos do conjunto de teste e não somente o de uma única classe, o valor de 0.99 é excelente. Junto com a média ponderada de 0.99 do *f1-score*, isto comprova a ótima eficácia do EFC em classificar os fluxos neste experimento.

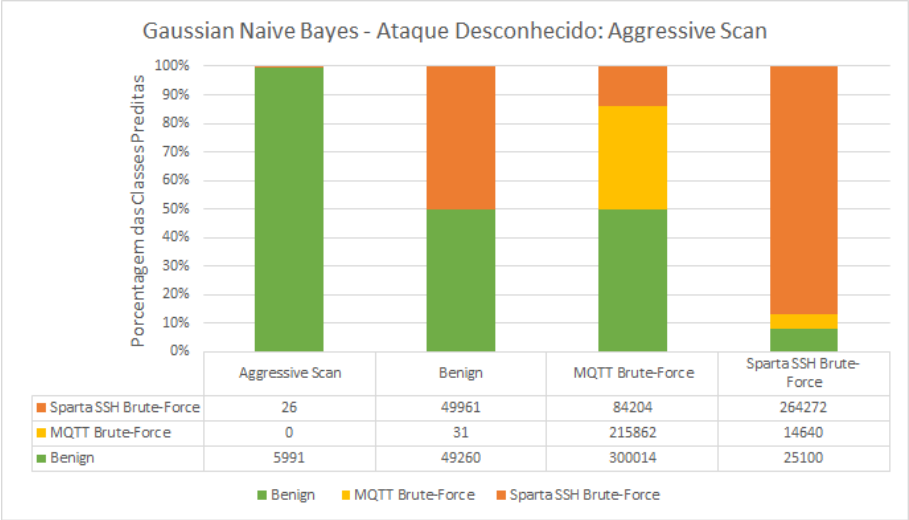

Figura 5.20: Gráfico de distribuição de classes do *Gaussian Naive Bayes*.

Tabela 5.14: Métricas de desempenho do *Gaussian Naive Bayes*

Labels	Precision	Recall	F1-Score
Benign	0.13	0.50	0.20
MQTT Brute-Force	0.94	0.36	0.52
Sparta SSH Brute-Force	0.66	0.87	0.75
Média Ponderada	0.52	0.53	0.53

Observando o gráfico da Figura 5.14, é fácil notar que o desempenho do *Gaussian Naive Bayes* não foi bom. Assim como o *Decision Tree* no gráfico da Figura 5.18, quase todos os ataques desconhecidos são classificados como fluxos benignos e juntamente com o fato de que aproximadamente metade dos fluxos de ataque *MQTT Brute-Force* foram classificados como benignos, ajuda a explicar o valor de 0.13 da *precision* da classe *Benign* na Tabela 5.14. Além disso, olhando para a barra de fluxos benignos, nota-se que aproximadamente metade dos fluxos são classificados como fluxos maliciosos, evidenciado pelo valor do *recall* de 0.50 da classe *Benign*. No contexto de uso para NIDSs, esta combinação de características estaria longe do ideal, o que é deixado claro pela média ponderada de somente 0.53 do *f1-score*.

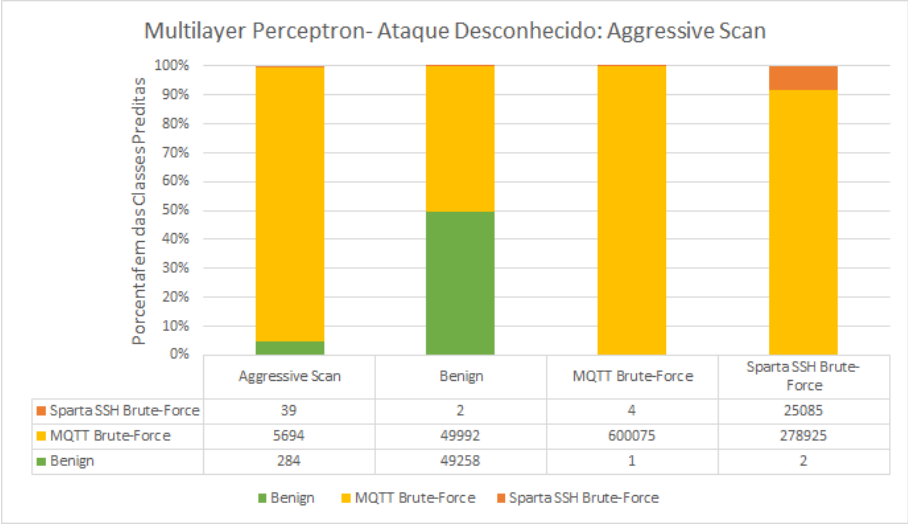

Figura 5.21: Gráfico de distribuição de classes do *Multilayer Perceptron*.

Tabela 5.15: Métricas de desempenho do *Multilayer Perceptron*

Labels	Precision	Recall	F1-Score
Benign	0.99	0.50	0.66
MQTT Brute-Force	0.65	0.99	0.79
Sparta SSH Brute-Force	0.99	0.08	0.15
Média Ponderada	0.67	0.67	0.67

A primeira coisa que se nota no gráfico da Figura 5.21 é que ele é dominado pela classe *MQTT Brute-Force*. O fato de boa parte dos ataques desconhecidos serem classificados como *MQTT Brute-Force* é desejável, pois no contexto de algoritmos que não possuem uma classe específica para fluxos desconhecidos como o EFC, o resultado ideal para estes fluxos é que eles sejam classificados como qualquer uma das classes de ataque. Entretanto, mais de 50% dos fluxos benignos e de 90% dos fluxos *Sparta SSH Brute-Force* também acabam sendo classificados como *MQTT Brute-Force*, o que explica os *recalls* de 0.50 e 0.08 das classes *Benign* e *Sparta SSH Brute-Force* respectivamente, presentes na Tabela 5.15. Tudo isso, aliado a uma média ponderada de 0.67 do *f1-score*, mostra resultados abaixo da média.

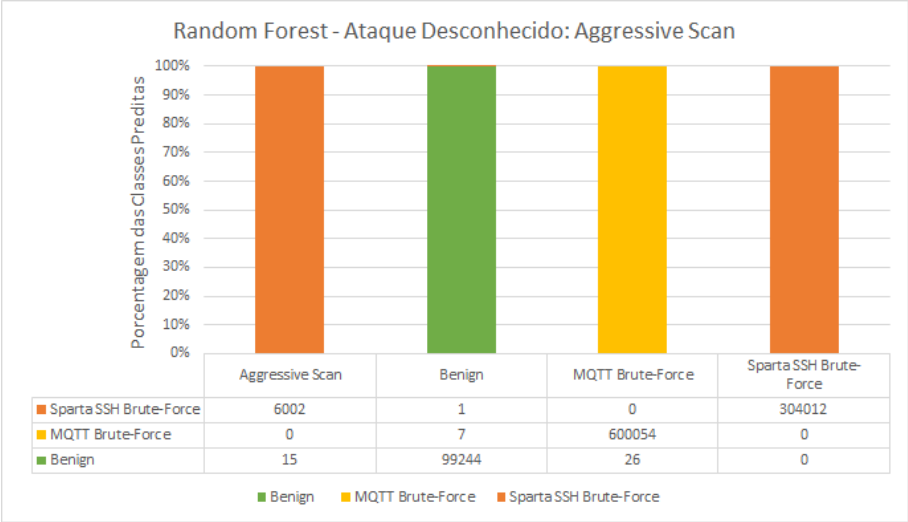

Figura 5.22: Gráfico de distribuição de classes do *Random Forest*.

Tabela 5.16: Métricas de desempenho do *Random Forest*

Labels	Precision	Recall	F1-Score
Benign	0.99	0.99	0.99
MQTT Brute-Force	0.99	0.99	0.99
Sparta SSH Brute-Force	0.99	1.00	0.99
Média Ponderada	0.99	0.99	0.99

O ótimo resultado do *Random Forest* neste experimento torna-se aparente logo de cara ao analisar o gráfico da Figura 5.22 e verificar praticamente todos os fluxos posicionados em suas colunas corretas. Vale notar que a grande maioria dos fluxos de ataque desconhecido foram classificados como um outro fluxo de ataque, aqui no caso como *Sparta SSH Brute-Force*. Este é inclusive o resultado ótimo, pois no contexto de algoritmos que não possuem uma classe específica para fluxos desconhecidos como o EFC, o resultado ideal é que eles sejam classificados como qualquer uma das classes de ataque. Este cenário aliado ao fato de que nenhuma métrica da Tabela 5.16 está abaixo de 0.99 faz com que o *Random Forest* tenha tido o melhor resultado da primeira fase do experimento, embora o *EFC* tenha desempenho comparável.

5.3.2 Fluxos *MQTT Brute-Force* como ataques desconhecidos

Nesta etapa do experimento, foram retirados os fluxos da classe *MQTT Brute-Force* do conjunto de treinamento para que servissem como fluxos desconhecidos e, a partir dos resultados, criados gráficos e tabelas para que se possa analisar o desempenho de cada algoritmo. A organização dos gráficos e das tabelas permanece a mesma utilizada na

Seção 5.3.1, com a diferença de que nesta etapa os algoritmos conseguem classificar os fluxos como *Aggressive Scan*, mas não como *MQTT Brute-Force*, que é o fluxo de ataque desconhecido.

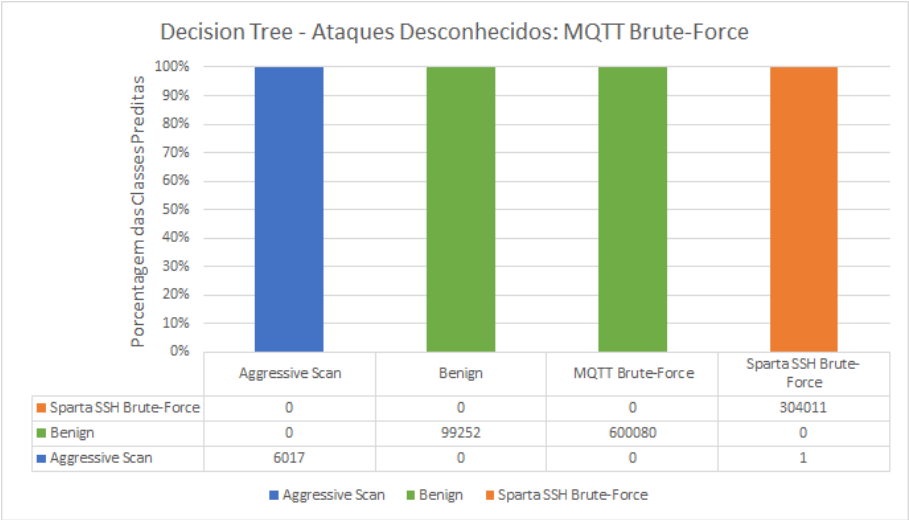

Figura 5.23: Gráfico de distribuição de classes do *Decision Tree*.

Tabela 5.17: Métricas de desempenho do *Decision Tree*

Labels	Precision	Recall	F1-Score
Aggressive Scan	0.99	1.00	0.99
Benign	0.14	1.00	0.25
Sparta SSH Brute-Force	1.00	0.99	0.99
Média Ponderada	0.40	0.99	0.58

Ao observar o gráfico da Figura 5.23, percebe-se desempenho semelhante ao do próprio algoritmo na Seção 5.3.1. Fluxos de classes conhecidas, em sua grande maioria, foram classificados corretamente, desempenho confirmado por nenhum valor de *recall* da Tabela 5.17 estar abaixo de 0.99. Já os fluxos de ataque desconhecidos, neste caso os de *MQTT Brute-Force*, foram quase em sua totalidade classificados como fluxos benignos, evidenciado pelo valor de 0.14 da *precision* da classe *Benign*. Entretanto, apesar das características semelhantes ao seu desempenho na Seção 5.3.1, o algoritmo *Decision Tree* possui um desempenho estatisticamente pior, como é possível ver com a média ponderada do *f1-score* de 0.58. Isso ocorre pois a classe *MQTT Brute-Force* que foi erroneamente classificada como benigna possui muito mais amostras do que a classe que anteriormente foi classificada errada, o que faz com que o impacto no desempenho geral seja muito maior.

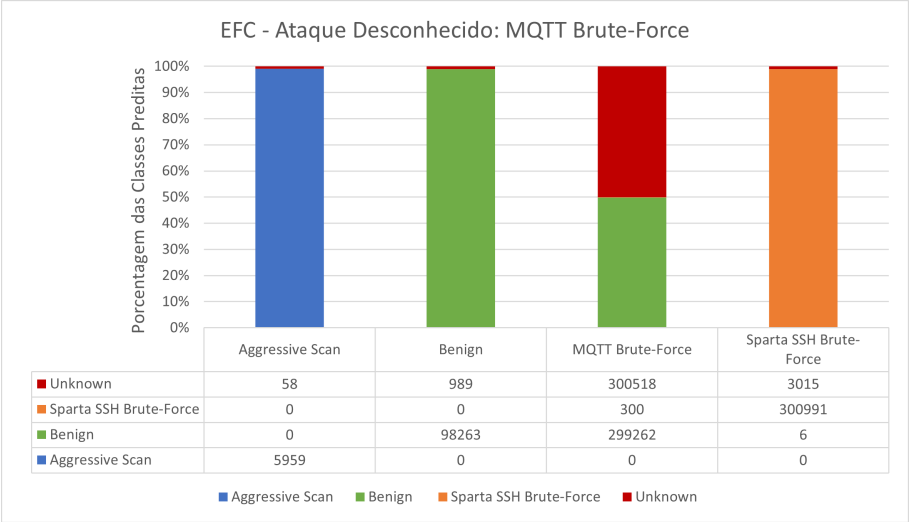

Figura 5.24: Gráfico de distribuição de classes do EFC.

Tabela 5.18: Métricas de desempenho do EFC

Labels	Precision	Recall	F1-Score
Aggressive Scan	0.99	0.99	0.99
Benign	0.25	0.99	0.40
Unknown	0.99	0.50	0.66
Sparta SSH Brute-Force	0.99	0.99	0.99
Média Ponderada	0.92	0.70	0.74

O gráfico da Figura 5.24 é organizado da mesma forma que o do EFC na Seção 5.3.1, com a diferença de que o algoritmo agora é capaz de classificar fluxos como pertencentes à classe *Aggressive Scan* e não mais como pertencentes à classe *MQTT Brute-Force*. Analisando este gráfico, percebe-se que, assim como na Seção 5.3.1, o EFC realizou um ótimo trabalho classificando os fluxos em que foi treinado, como evidenciado pelo valor de 0.99 do *recall* das classes *Aggressive Scan*, *Benign* e *Sparta SSH Brute-Force* presentes na Tabela 5.18. Entretanto, ao se analisar a barra da classe *MQTT Brute-Force*, somente pouco mais da metade dos fluxos foram corretamente classificados como *Unknown*, enquanto todo o resto foi classificado como *Benign*, resultados que explicam os valores da *precision* de 0.25 da classe *Benign* e do *recall* de 0.50 da classe *Unknown*. Essa classificação incorreta de metade fluxos de ataque desconhecidos faz com que a média ponderada do *f1-score* do EFC seja somente 0.74, evidenciando seu desempenho ruim se comparado ao valor de 0.99 que atingiu no experimento da Seção 5.3.1.

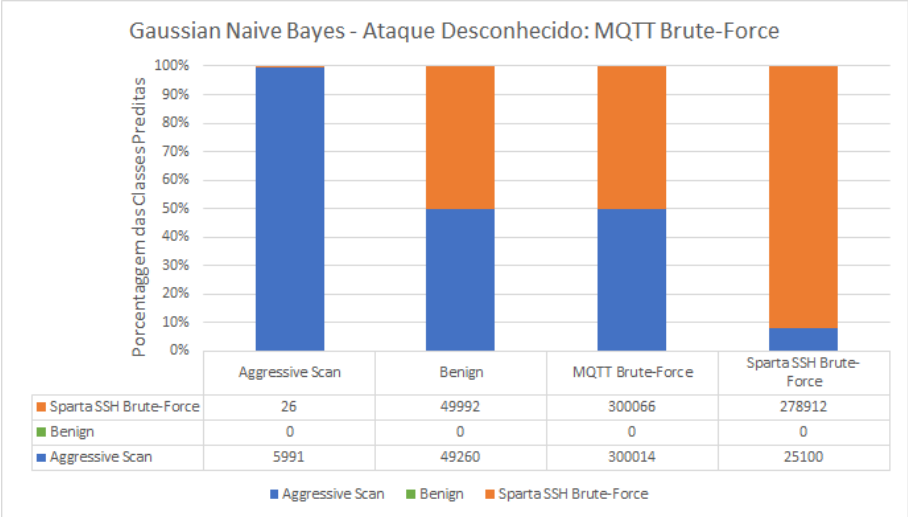

Figura 5.25: Gráfico de distribuição de classes do *Gaussian Naive Bayes*.

Tabela 5.19: Métricas de desempenho do *Gaussian Naive Bayes*

Labels	Precision	Recall	F1-Score
Aggressive Scan	0.80	0.99	0.89
Benign	0.00	0.00	0.00
Sparta SSH Brute-Force	0.92	0.92	0.92
Média Ponderada	0.88	0.70	0.85

Ao analisar o gráfico da Figura 5.25, nota-se que há somente duas cores preenchendo as barras, a das classes *Aggressive Scan* e *Sparta SSH Brute-Force*. Isso significa que nenhum dos fluxos do conjunto de teste foi classificado como benigno, evidenciado por todas das métricas da classe *Benign* na Tabela 5.19 possuírem valor 0.00. Entretanto, o valor de 0.85 da média ponderada do *f1-score* não é baixo. Isso ocorre por que, como nenhum fluxo é classificado como benigno, necessariamente todos os fluxos de ataque desconhecidos estão corretamente classificados. Como explicado anteriormente, todo fluxo de ataque desconhecido que for classificado como qualquer um dos ataques conhecidos é considerado correto para todo algoritmo que não seja o EFC. E já que os fluxos de ataque desconhecidos compõem cerca de 60% de todas as amostras do conjunto de testes nesta etapa do experimento, a sua taxa de acerto perfeita faz com que as métricas fiquem em valores muito altos, apesar do desempenho real não ser bom.

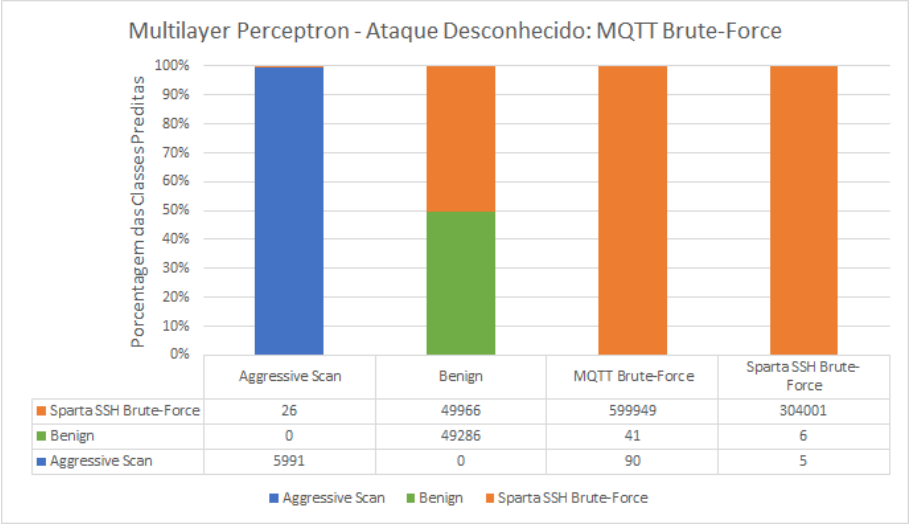

Figura 5.26: Gráfico de distribuição de classes do *Multilayer Perceptron*.

Tabela 5.20: Métricas de desempenho do *Multilayer Perceptron*

Labels	**Precision**	**Recall**	**F1-Score**
Aggressive Scan	0.99	0.99	0.99
Benign	0.99	0.50	0.66
Sparta SSH Brute-Force	0.95	0.99	0.97
Média Ponderada	0.95	0.88	0.94

Analisando o gráfico da Figura 5.26 é possível perceber que o *Multilayer Perceptron* obteve bom desempenho classificando corretamente os fluxo de ataque, como confirma o valor de 0.99 no *recall* das classes *Aggressive Scan* e *Sparta SSH Brute-Force* na Tabela 5.20. Isso inclui os fluxos de ataque desconhecidos, nesta etapa do experimento a classe *MQTT Brute-Force*, que foram em sua grande maioria classificados como *Sparta SSH Brute-Force*, o que é correto segundo o critério estabelecido na introdução da Seção 5.3. Entretanto, o desempenho ao classificar os fluxos benignos não foi muito bom, onde pouco mais da metade deles foram classificados incorretamente como fluxos *Sparta SSH Brute-Force*, situação evidenciada na Tabela 5.20 pelo *recall* de 0.50 da classe *Benign* e *precision* de 0.95 da classe *Sparta SSH Brute-Force*. Embora a média ponderada do *f1-score* seja alta no valor de 0.94, o fato de classificar erroneamente metade dos fluxos benignos faz com que seu desempenho real não seja muito bom.

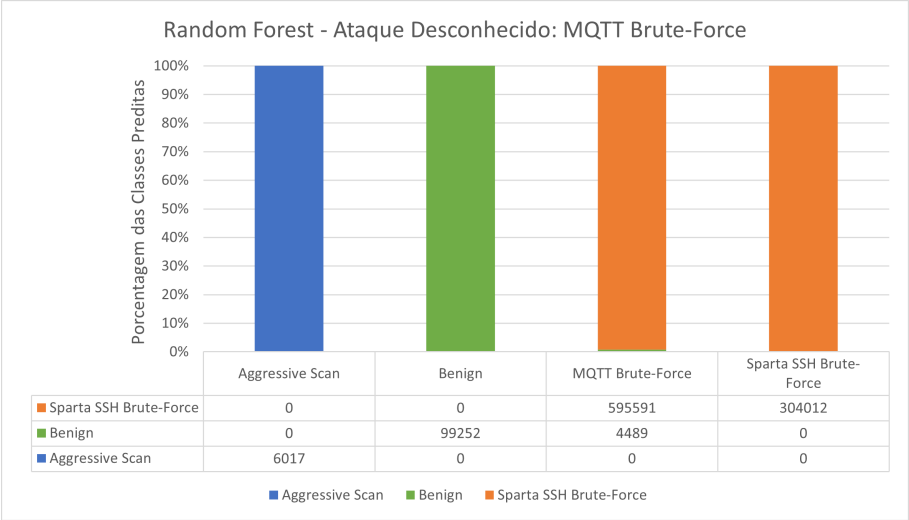

Figura 5.27: Gráfico de distribuição de classes do *Random Forest*.

Tabela 5.21: Métricas de desempenho do *Random Forest*

Labels	Precision	Recall	F1-Score
Aggressive Scan	1.00	1.00	1.00
Benign	0.96	1.00	0.98
Sparta SSH Brute-Force	1.00	1.00	1.00
Média Ponderada	0.99	1.00	0.99

Por fim, analisando o gráfico da Figura 5.27, nota-se que os fluxos de ataque conhecidos foram perfeitamente classificados, fato confirmado na Tabela 5.21 pela *precision* e *recall* de valor 1.00 tanto na classe *Aggressive Scan* quanto na classe *Sparta SSH Brute-Force*. A única coisa que impediu o *Random Forest* de receber uma avaliação perfeita neste experimento foi que uma pequena parte dos fluxos de ataque desconhecidos foram classificados como fluxos benignos, como evidenciado pela *precision* de 0.96 da classe *Benign*. Com média ponderada do *f1-score* com valor 0.99, é possível concluir que o *Random Forest* obteve um excelente desempenho nesta etapa do experimento, o melhor entre todos os algoritmos.

5.3.3 Fluxos *Sparta SSH Brute-Force* como ataques desconhecidos

Nesta etapa do experimento, foram retirados os fluxos da classe *Sparta SSH Brute-Force* do conjunto de treinamento para que servissem como fluxos desconhecidos e, a partir dos resultados, criados gráficos e tabelas para que se possa analisar o desempenho de cada algoritmo. A organização dos gráficos e das tabelas permanece a mesma dos da Seção

5.3.2, com a diferença de que nesta etapa os algoritmos conseguem classificar os fluxos como *MQTT Brute-Force*, mas não como *Sparta SSH Brute-Force*, que é o fluxo de ataque desconhecido.

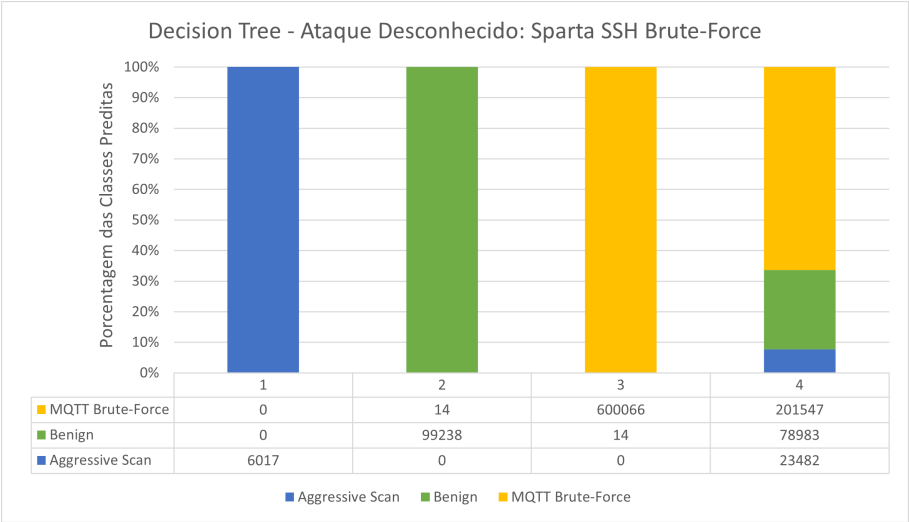

Figura 5.28: Gráfico de distribuição de classes do *Decision Tree*.

Tabela 5.22: Métricas de desempenho do *Decision Tree*

Labels	Precision	Recall	F1-Score
Aggressive Scan	1.00	1.00	1.00
Benign	0.56	0.99	0.71
MQTT Brute-Force	0.99	0.99	0.99
Média Ponderada	0.92	0.99	0.95

Observando o gráfico da Figura 5.28, o algoritmo *Decision Tree* classificou corretamente a grande maioria dos fluxos com o qual treinou, desempenho confirmado na Tabela 5.22 com os valores de *recall* sendo pelo menos 0.99 em cada uma das classes. Entretanto, ao classificar os fluxos de ataque desconhecidos, cerca de 25% dos fluxos foram classificados como benignos, comportamento que explica a *precision* baixa de 0.56 da classe *Benign*. Embora o valor da média ponderada do *f1-score* de 0.95 seja boa, o fato de que quase metade dos fluxos classificados como benignos são falsos positivos faz com que seu desempenho real não seja tão alto assim.

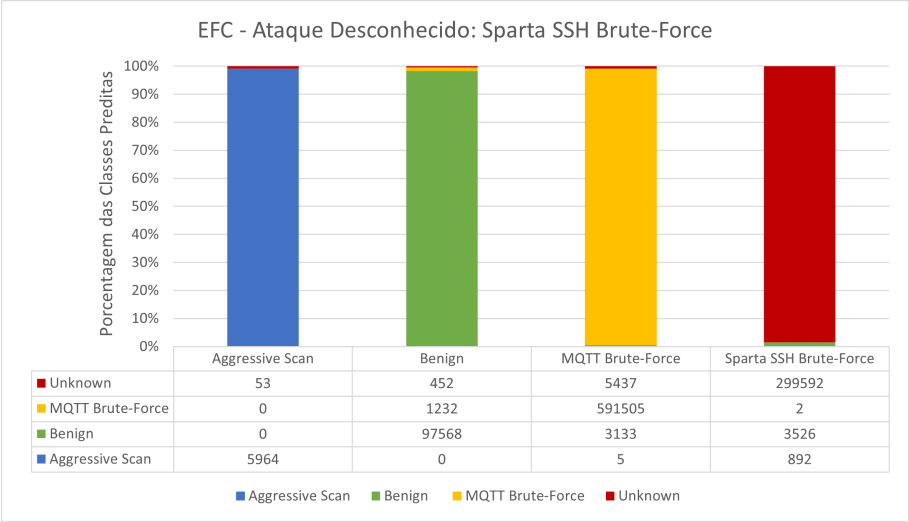

Figura 5.29: Gráfico de distribuição de classes do EFC.

Tabela 5.23: Métricas de desempenho do EFC

Labels	Precision	Recall	F1-Score
Aggressive Scan	0.87	0.99	0.93
Benign	0.94	0.98	0.96
MQTT Brute-Force	0.99	0.99	0.99
Unknown	0.98	0.99	0.98
Média Ponderada	0.99	0.99	0.99

O gráfico da Figura 5.29 é organizado da mesma forma que o do EFC na Seção 5.3.2, com a diferença de que o algoritmo agora é capaz de classificar fluxos como pertencentes à classe *MQTT Brute-Force* e não mais como pertencentes à classe *Sparta SSH Brute-Force*. Analisando este gráfico, percebe-se que, de modo geral, o EFC realizou um ótimo trabalho tanto na classificação de fluxos com o qual treinou quanto com os fluxos de ataque desconhecidos, desempenho que pode ser confirmado na Tabela 5.23, visto que nenhum *recall* é menor do que 0.98.

Nota-se que os valores da *precision* das classes *Aggressive Scan* e *Benign* estão um pouco abaixo da média das outras métricas. Isso ocorre pois as duas classes têm o menor número de amostras no conjunto de teste, especialmente a classe *Aggressive Scan*. Portanto, mesmo que uma pequena porcentagem de fluxos seja erroneamente classificada, é o suficiente para que a *precision* ddelas seja significativamente impactada. Apesar disso, com média ponderada do f1-score de 0.99, o desempenho do EFC é excelente, o melhor nesta etapa dos experimentos.

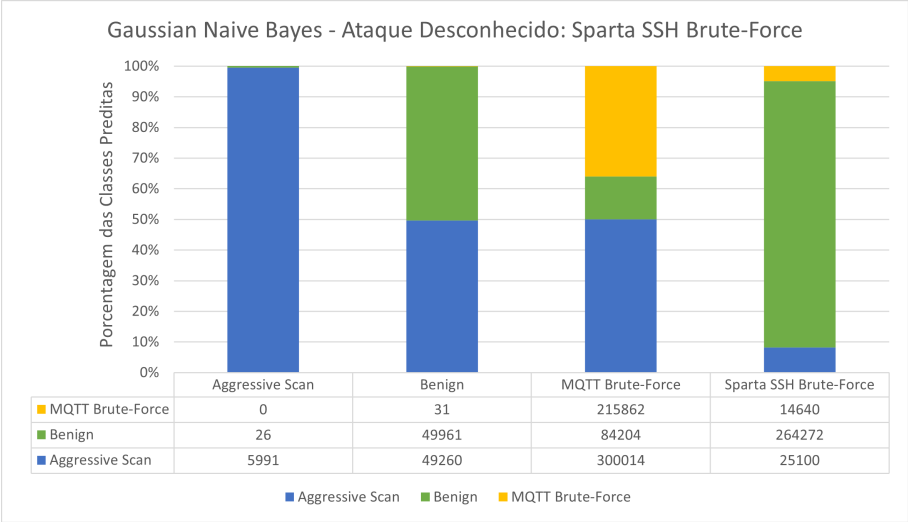

Figura 5.30: Gráfico de distribuição de classes do *Gaussian Naive Bayes*.

Tabela 5.24: Métricas de desempenho do *Gaussian Naive Bayes*

Labels	Precision	Recall	F1-Score
Aggressive Scan	0.08	0.99	0.15
Benign	0.12	0.50	0.20
MQTT Brute-Force	0.99	0.36	0.53
Média Ponderada	0.31	0.38	0.35

Ao analisar o gráfico da Figura 5.30, percebe-se que a única classe de fluxos no conjunto de teste que foi classificada corretamente quase por inteiro foi a *Aggressive Scan*, como evidenciado pelo *recall* de 0.99 na Tabela 5.24. De resto, há vários problemas no desempenho do *Gaussian Naive Bayes*. Mais da metade dos fluxos benignos foram classificados como *Aggressive Scan*, evidenciado pelo *recall* de 0.50 da classe *Benign*. Além disso, cerca de 15% dos fluxos *MQTT Brute-Force* e 90% dos fluxos de ataque desconhecidos foram classificados como fluxos benignos, o que também ajuda a explicar o valor de 0.12 da *precision* da classe *Benign*. Tudo isso, aliado ao valor da média ponderada do *f1-score* de 0.35, faz com que este seja o pior resultado de um classificador em todo o conjunto de experimentos.

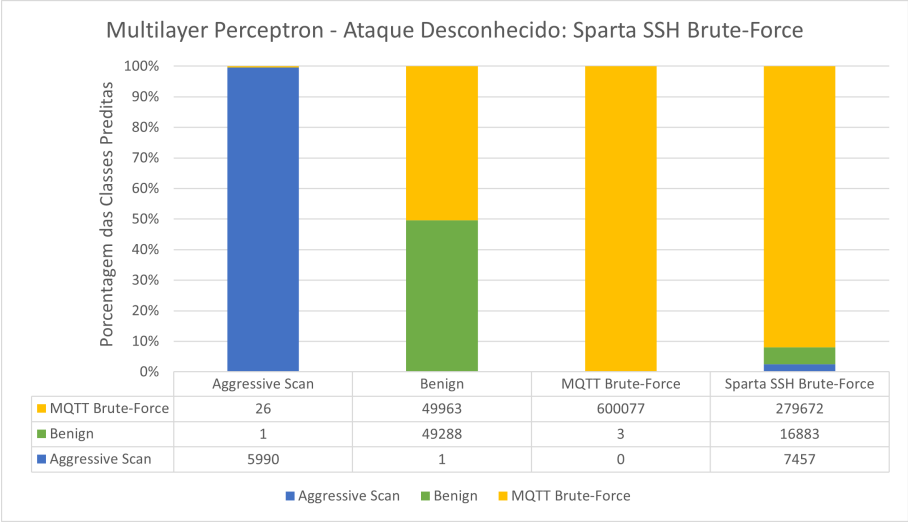

Figura 5.31: Gráfico de distribuição de classes do *Multilayer Perceptron*.

Tabela 5.25: Métricas de desempenho do *Multilayer Perceptron*

Labels	Precision	Recall	F1-Score
Aggressive Scan	0.99	0.99	0.99
Benign	0.74	0.50	0.60
MQTT Brute-Force	0.95	0.99	0.97
Média Ponderada	0.93	0.93	0.94

Observando o gráfico da Figura 5.31, é possível notar que o *Multilayer Perceptron* classificou corretamente quase todos os fluxos pertencentes às duas classes de ataque com qual o algoritmo foi treinado, como explicitado na Tabela 5.25 pelo valor do *recall* de 0.99 nas classes *Aggressive Scan* e *MQTT Brute-Force*. Já os fluxos de ataque desconhecidos, nesta presente etapa do experimento os fluxos *Sparta SSH Brute-Force*, foram em sua maioria classificados como *MQTT Brute-Force*, o que para o contexto deste experimento é o desejável, uma vez que o *Multilayer Perceptron* não possui uma classe específica para fluxos de ataque desconhecidos. Entretanto, uma parte significativa dos fluxos de ataque desconhecidos foram erroneamente classificados como benignos, o que, devido à quantidade elevada de amostras em comparação aos fluxos benignos, impacta na *precision* da classe *Benign* e faz com que fique em 0.74.

O verdadeiro problema dos resultados tem a ver com a classificação dos fluxos benignos. Assim como em suas avaliações nas subseções 5.3.1 e 5.3.2, o algoritmo *Multilayer Perceptron* teve dificuldades em classificar estes fluxos, sendo que aproximadamente 50% deles foram classificados erroneamente como *Aggressive Scan*, fato corroborado pelo valor de 0.50 do *recall* da classe *Benign* na Tabela 5.25. Portanto, apesar da média ponde-

rada do *f1-score* ser de 0.94, a dificuldade em classificar fluxos benignos faz com que seu desempenho real não seja tão bom quanto o *f1-score* indica.

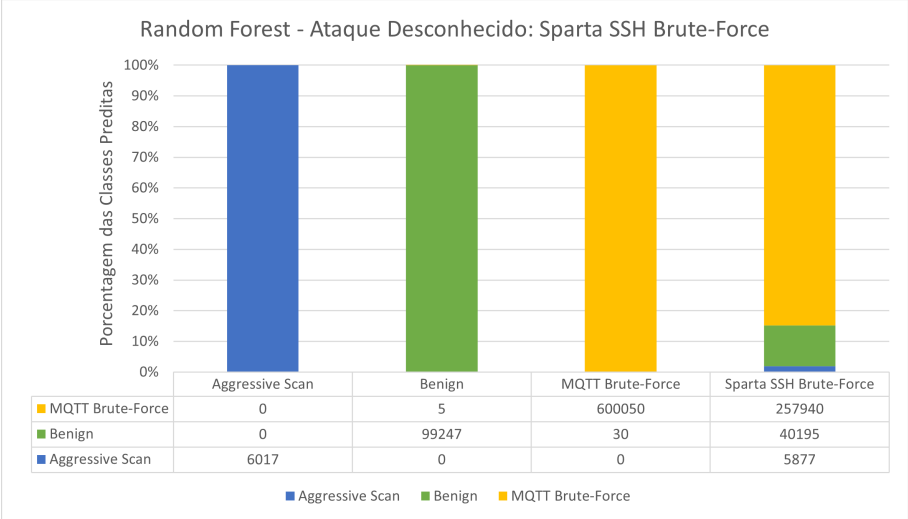

Figura 5.32: Gráfico de distribuição de classes do *Random Forest*.

Tabela 5.26: Métricas de desempenho do *Random Forest*

Labels	Precision	Recall	F1-Score
Aggressive Scan	1.00	1.00	1.00
Benign	0.71	0.99	0.83
MQTT Brute-Force	0.99	0.99	0.99
Média Ponderada	0.96	0.99	0.97

Por fim, ao se analisar o gráfico da Figura 5.32, é possível perceber que o *Random Forest* classificou corretamente a grande maioria dos fluxos com que treinou, fato confirmado pela Tabela 5.26, tendo em vista que nenhum valor de *recall* é menor do que 0.99. Entretanto, cerca de 15% dos fluxos de ataque desconhecidos foram classificados como fluxos benignos, o que faz com que a *precision* da classe *Benign* assuma valor de 0.71, o pior valor entre todas as métricas do *Random Forest*, tanto nesta seção quanto na Seção 5.3.1 e Seção 5.3.2. Apesar deste problema, com a média ponderada do *f1-score* sendo 0.97, o *Random Forest* possui um bom desempenho, embora nesta etapa do experimento o EFC leve vantagem.

5.3.4 Sintetização dos Resultados

Nesta seção será apresentada uma tabela comparando o desempenho dos algoritmos nos experimentos e, a partir dela, serão analisados os resultados de cada um. A métrica

escolhida foi a média ponderada do *f1-score*, pois leva em conta tanto as outras métricas *recall* e *precision* quanto o peso da quantidade de fluxos que cada classe possui no conjunto de testes. Os valores em amarelo são os que possuíram melhor desempenho em seu respectivo experimento.

Tabela 5.27: Tabela de comparação da média ponderada do f1-score de cada algoritmo.

	Aggressive Scan	**MQTT BF**	**Sparta BF**
Decision Tree	0.99	0.58	0.95
EFC	0.99	0.74	0.99
Gaussian Naive Bayes	0.53	0.85	0.35
Multilayer Perceptron	0.67	0.94	0.94
Random Forest	0.99	0.99	0.97

Na primeira etapa do experimento, apresentada na Seção 5.3.1, foram retirados todos os fluxos da classe *Aggressive Scan* do conjunto de treinamento para que quando aparecessem no conjunto de teste, agissem como um fluxo de ataque desconhecido. Observando a Tabela 5.27 na coluna *Aggressive Scan*, dois algoritmos foram apontados como tendo o melhor desempenho, o EFC e o *Random Forest*, com a média ponderada do *f1-score* de 0.99. Ambos classificaram corretamente a grande maioria dos fluxos de cada classe, como é possível ver pelas barras de cor quase uniforme nos gráficos da Figura 5.19 e Figura 5.22.

Nessa mesma coluna é possível ver que o *Decision Tree* também possui a média ponderada do *f1-score* de 0.99 mas não foi marcado como um dos algoritmos com melhor desempenho. Isso ocorre pois, como é possível ver no gráfico da Figura 5.18, quase todos os fluxos desconhecidos foram classificados erroneamente como benignos, o que no contexto de uso para NIDSs é um dos piores cenários possíveis. O motivo da média ponderada do *f1-score* continuar alta é que a quantidade de fluxos de ataque desconhecidos é muito pequena se comparada com a quantidade total de fluxos e, portanto, não afeta tanto a métrica.

Na segunda etapa do experimento, apresentada na Seção 5.3.2, foram retirados todos os fluxos da classe *MQTT Brute-Force* do conjunto de treinamento para que quando aparecessem no conjunto de teste agissem como um fluxo de ataque desconhecido. Observando a Tabela 5.27, na coluna *MQTT Brute-Force*, que representa essa etapa do experimento, o algoritmo *Random Forest* mostrou melhor desempenho, como evidenciado pela média ponderada do *f1-score* de 0.99. Ele classificou corretamente a grande maioria dos fluxos de cada classe, como é possível ver pelas barras de cor quase uniforme no gráfico da Figura 5.27.

Já o EFC, o algoritmo de maior interesse para este trabalho, teve o seu pior desempenho nesta etapa do experimento. Analisando o gráfico da Figura 5.24, é possível perceber que ele analisou de forma correta quase todos os fluxos com que treinou, mas acabou clas-

sificando metade dos fluxos de ataque desconhecidos como fluxos benignos. Conjectura-se que este desempenho abaixo da média tenha sido causado pelo desbalanceamento do *dataset*, uma vez que, nesta etapa do experimento, os fluxos de ataque desconhecidos correspondem a quase dois terços de todos os fluxos do conjunto de teste. Essa disparidade também faz com que a média ponderada do *f1-score* do EFC seja consideravelmente menor do que se ele tivesse classificado erroneamente outra classe de ataque.

Na terceira e última etapa do experimento, apresentada na Seção 5.3.3, foram retirados todos os fluxos da classe *Sparta SSH Brute-Force* do conjunto de treinamento para que quando aparecessem no conjunto de teste agissem como um fluxo de ataque desconhecido. Observando a Tabela 5.27 na coluna *Sparta SSH Brute-Force*, que representa esta etapa do experimento, o algoritmo EFC mostrou o melhor desempenho, como evidenciado pela média ponderada do *f1-score* de 0.99. Ele classificou corretamente a grande maioria dos fluxos de cada classe, como é possível ver pelas barras de cor quase uniforme nos gráficos da Figura 5.29. É seguido de perto pelo *Random Forest*, que apesar de possuir uma boa média ponderada do *f1-score* de 0.97, possui uma quantidade significativa de fluxos de ataque desconhecidos classificados como fluxos benignos.

Capítulo 6

Conclusão

Neste trabalho, propôs-se a utilização do EFC para detecção de intrusões de rede em um contexto de IoT utilizando o protocolo MQTT. Na etapa de treino do EFC binário, que consegue classificar os fluxos em benignos ou maliciosos, ele infere um modelo estatístico baseado no modelo de Potts, tendo como entrada somente amostras de tráfego benigno. Em seguida, este modelo estatístico é utilizado para classificar fluxos de rede entre benignos ou maliciosos com base em valores de energia. Os resultados obtidos na Seção 5.1, com valores de *f1-score* chegando a 1.0, mostram que o *EFC* binário é capaz de classificar corretamente a grande maioria dos fluxos do *dataset* MQTT-IoT-IDS2020 utilizado neste trabalho.

Foi testada também a versão multi-classe do EFC, na qual em vez de classificar somente fluxos em benignos ou maliciosos, ele é capaz de identificar os tipos de cada ataque com que foi treinado. Para isso, o algoritmo utiliza um EFC binário para inferir um modelo estatístico para cada classe de fluxo presente no conjunto de treino. Em seguida, os modelos estatísticos são utilizados para classificar os fluxos de rede, com base no valor da menor energia que for retornada. Os resultados obtidos na Seção 5.2 mostram que o *EFC* multi-classe é capaz de classificar corretamente a grande maioria dos fluxos do *dataset* MQTT-IoT-IDS2020 utilizado nesse trabalho. Os valor do *f1-score* de 0.99 do EFC é maior ou comparável ao valor desta mesma métrica obtida pelos melhores algoritmos de aprendizado de máquina supervisionado tradicionais, como o *Random Forest* e o *Multilayer Perceptron*.

Além disso, para melhor testar a viabilidade da aplicação do EFC multi-classe para NIDSs, foi analisada a capacidade do algoritmo de lidar com fluxos de ataque desconhecidos. Para isso, foram retirados todos os fluxos de uma classe de ataque do conjunto de treinamento, mas deixados no conjunto de teste. Os resultados obtidos na Seção 5.3 mostram que o *EFC* multi-classe foi eficiente ao classificar corretamente os fluxos, mesmo com a presença de fluxos desconhecidos. Em dois dos três cenários de teste, o valor do

f1-score de 0.99 do EFC é maior ou comparável ao valor da mesma métrica obtida pelos melhores classificadores de Machine Learning da literatura clássica, como o *Random Forest* e o *Multilayer Perceptron*. Entretanto, no cenário do experimento da Seção 5.3.2, o EFC mostrou resultados abaixo da média, com *f1-score* de 0.72. Conjectura-se que o desempenho abaixo da média tenha sido causado pelo desbalanceamento do *dataset*, uma vez que, neste cenário, os fluxos de ataques desconhecidos correspondiam a quase dois terços de todos os fluxos do conjunto de teste.

Tendo em vista os resultados discutidos no capítulo anterior, há fortes evidências de que o EFC é um algoritmo promissor para utilização em NIDSs em um contexto IoT MQTT, com potencial para uso mais amplo. Entretanto, há espaço para mais testes sobre sua aplicação na área. Seria interessante em trabalhos futuros encontrar mais *datasets* de fluxos IoT MQTT, inclusive se possível de tráfegos não-simulados, para testar a adaptabilidade do algoritmo a diferentes domínios. Também seria interessante explorar a possibilidade de realizar *feature selecting* sobre os fluxos e de como incorporar o EFC em um NIDS.

Referências

[1] Demeter, Dan: *Kaspersky: Attacks on iot devices double in a year.* `https://iottechnews.com/news/2021/sep/07/kaspersky-attacks-on-iot-devices-double-in-a-year/`, acesso em 05-11-2021. 1

[2] ENISA: *ENISA Threat Landscape.* Relatório Técnico, European Union Agency for Cybersecurity (ENISA), 2020. 1

[3] DCMS: *Cyber Security Breaches Survey 2021.* Relatório Técnico, United Kingdom's Department for Digital, Culture, Media and Sport (DCMS), 2021. 1

[4] ACSC: *ACSC Annual Cyber Threat Report.* Relatório Técnico, Australian Cyber Security Center (ACSC), 2021. 1

[5] Sethi, Pallavi e Smruti R Sarangi: *Internet of things: architectures, protocols, and applications.* Journal of Electrical and Computer Engineering, 2017, 2017. 1

[6] Kiran, KVVNL Sai, RN Kamakshi Devisetty, N Pavan Kalyan, K Mukundini e R Karthi: *Building a intrusion detection system for iot environment using machine learning techniques.* Procedia Computer Science, 171:2372–2379, 2020. 1, 10, 11, 17

[7] Hasan, Mahmudul, Md Milon Islam, Md Ishrak Islam Zarif e MMA Hashem: *Attack and anomaly detection in iot sensors in iot sites using machine learning approaches.* Internet of Things, 7:100059, 2019. 1, 10, 17

[8] Khraisat, Ansam, Iqbal Gondal, Peter Vamplew, Joarder Kamruzzaman e Ammar Alazab: *A novel ensemble of hybrid intrusion detection system for detecting internet of things attacks.* Electronics, 8(11):1210, 2019. 1

[9] Midi, Daniele, Antonino Rullo, Anand Mudgerikar e Elisa Bertino: *Kalis—a system for knowledge-driven adaptable intrusion detection for the internet of things.* Em *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, páginas 656–666. IEEE, 2017. 1

[10] Hindy, Hanan, Ethan Bayne, Miroslav Bures, Robert Atkinson, Christos Tachtatzis e Xavier Bellekens: *Machine learning based iot intrusion detection system: an mqtt case study (mqtt-iot-ids2020 dataset).* Em *International Networking Conference*, páginas 73–84. Springer, 2020. 2, 8, 10, 14, 15, 16

[11] Pontes, Camila, Manuela Souza, João Gondim, Matt Bishop e Marcelo Marotta: *A new method for flow-based network intrusion detection using the inverse potts model.*

IEEE Transactions on Network and Service Management, 2021. 2, 4, 10, 12, 13, 15, 16, 19, 20

[12] Pontes, Camila, Manuela Souza, João Gondim, Luís P.F. Garcia, Marcelo Marotta e Luiz DaSilva: *Um novo método baseado no modelo de potts para detecção de intrusão em rede.* Monografia UnB, 2020. 2

[13] Pontes, Camila FT, Manuela MC de Souza, João JC Gondim, Matt Bishop e Marcelo Antonio Marotta: *A new method for flow-based network intrusion detection using the inverse potts model.* IEEE Transactions on Network and Service Management, 18(2):1125–1136, 2021. 2, 5, 6, 10, 13, 15, 16, 20, 23

[14] Buczak, Anna L e Erhan Guven: *A survey of data mining and machine learning methods for cyber security intrusion detection.* IEEE Communications surveys & tutorials, 18(2):1153–1176, 2015. 2, 19

[15] Ring, Markus, Sarah Wunderlich, Dominik Grüdl, Dieter Landes e Andreas Hotho: *Flow-based benchmark data sets for intrusion detection.* Em *Proceedings of the 16th European Conference on Cyber Warfare and Security. ACPI*, páginas 361–369, 2017. 2

[16] Sharafaldin, Iman, Arash Habibi Lashkari e Ali A Ghorbani: *Toward generating a new intrusion detection dataset and intrusion traffic characterization.* ICISSp, 1:108–116, 2018. 2

[17] Cocco, Simona, Christoph Feinauer, Matteo Figliuzzi, Rémi Monasson e Martin Weigt: *Inverse statistical physics of protein sequences: a key issues review.* Reports on Progress in Physics, 81(3):032601, 2018. 3

[18] Wu, Fa Yueh: *The potts model.* Reviews of modern physics, 54(1):235, 1982. 3, 12

[19] Stanford-Clark, Andy e Arlen Nipper: *Mq telemetry transport.* `https://mqtt.org/`, acesso em 20-04-2022. 6

[20] Eugster, Patrick Th, Pascal A Felber, Rachid Guerraoui e Anne Marie Kermarrec: *The many faces of publish/subscribe.* ACM computing surveys (CSUR), 35(2):114–131, 2003. 7

[21] Hanes, David, Gonzalo Salgueiro, Patrick Grossetete, Robert Barton e Jerome Henry: *IoT fundamentals: Networking technologies, protocols, and use cases for the internet of things.* Cisco Press, 2017. 7

[22] McCanne, Steve, Craig Leres e Van Jacobson: *tcpdump.* `https://www.tcpdump.org/`, acesso em 20-04-2022. 9

[23] Arkin, Ofir: *Network scanning techniques.* Publicom Communications Solutions, 1999. 9

[24] Durumeric, Zakir, Eric Wustrow e J Alex Halderman: *Fast internet-wide scanning and its security applications.* Em *22nd USENIX Security Symposium (USENIX Security 13)*, páginas 605–620, 2013. 9

[25] Bou-Harb, Elias, Mourad Debbabi e Chadi Assi: *Cyber scanning: a comprehensive survey.* Ieee communications surveys & tutorials, 16(3):1496–1519, 2013. 9

[26] Lyon, Gordon "Fyodor": *Nmap.* `https://nmap.org/`, acesso em 20-04-2022. 9

[27] Hossain, Md Delwar, Hideya Ochiai, Fall Doudou e Youki Kadobayashi: *Ssh and ftp brute-force attacks detection in computer networks: Lstm and machine learning approaches.* Em *2020 5th International Conference on Computer and Communication Systems (ICCCS)*, páginas 491–497. IEEE, 2020. 9

[28] Stiawan, Deris, Mohd Idris, Reza Firsandaya Malik, Siti Nurmaini, Nizar Alsharif, Rahmat Budiarto *et al.*: *Investigating brute force attack patterns in iot network.* Journal of Electrical and Computer Engineering, Vol. 2019, 2019. 9

[29] Quina, Antonio e Leonidas Stavliotis: *Sparta.* `https://github.com/SECFORCE/sparta`, acesso em 20-04-2022. 9

[30] Abeles, Daniel e Moshe Zioni: *Mqtt-pwn.* `https://mqtt-pwn.readthedocs.io/en/latest/`, acesso em 20-04-2022. 9

[31] Aubet, FX e MO Pahl: *Ds2os traffic traces*, 2018. 11

[32] Habibi Lashkari, Arash: *Cicflowmeter.* `https://github.com/ahlashkari/CICFlowMeter`, acesso em 20-04-2022. 15

[33] Cournapeau, David: *scikit-learn.* `https://scikit-learn.org`, acesso em 20-04-2022. 17

[34] Cournapeau, David: *Svc in scikit-learn.* `https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`, acesso em 20-04-2022. 17

[35] Webb, Geoffrey I, Eamonn Keogh e Risto Miikkulainen: *Naïve bayes.* Encyclopedia of machine learning, 15:713–714, 2010. 17

[36] Russell, Stuart e Peter Norvig: *Artificial intelligence: a modern approach.* Prentice Hall, 2010. 17

[37] Quinlan, J. Ross: *Simplifying decision trees.* International journal of man-machine studies, 27(3):221–234, 1987. 17

[38] Stein, Gary, Bing Chen, Annie S Wu e Kien A Hua: *Decision tree classifier for network intrusion detection with ga-based feature selection.* Em *Proceedings of the 43rd annual Southeast regional conference-Volume 2*, páginas 136–141, 2005. 17

[39] Yuan, Ye, Liji Wu e Xiangmin Zhang: *Gini-impurity index analysis.* IEEE Transactions on Information Forensics and Security, 16:3154–3169, 2021. 17

[40] Tangirala, Suryakanthi: *Evaluating the impact of gini index and information gain on classification using decision tree classifier algorithm.* International Journal of Advanced Computer Science and Applications, 11(2):612–619, 2020. 17

[41] Rigatti, Steven J: *Random forest.* Journal of Insurance Medicine, 47(1):31–39, 2017. 18

[42] Gupta, Govind P e Manish Kulariya: *A framework for fast and efficient cyber security network intrusion detection using apache spark.* Procedia Computer Science, 93:824–831, 2016. 18

[43] Krogh, Anders: *What are artificial neural networks?* Nature biotechnology, 26(2):195–197, 2008. 18

[44] Shenfield, Alex, David Day e Aladdin Ayesh: *Intelligent intrusion detection systems using artificial neural networks.* Ict Express, 4(2):95–99, 2018. 18

[45] Hancock, John T e Taghi M Khoshgoftaar: *Survey on categorical data for neural networks.* Journal of Big Data, 7(1):1–41, 2020. 19

[46] Cournapeau, David: *Labelencoder in scikit-learn.* `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html`, acesso em 20-04-2022. 19

[47] Cournapeau, David: *classification_report in scikit-learn.* `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html`, acesso em 20-04-2022. 20, 21

[48] Grandini, Margherita, Enrico Bagli e Giorgio Visani: *Metrics for multi-class classification: an overview.* arXiv preprint arXiv:2008.05756, 2020. 20, 22

[49] Pontes, Camila, Manuela Souza, João Gondim, Matt Bishop e Marcelo Marotta: *Implementation of efc.* `https://efc-package.readthedocs.io/en/latest/index.html`, acesso em 20-04-2022. 22, 23

[50] Sahare, Mahendra e Hitesh Gupta: *A review of multi-class classification for imbalanced data.* International Journal of Advanced Computer Research, 2(3):160, 2012. 22

[51] Management, AQR Capital: *Website of pandas.* `https://pandas.pydata.org/`, acesso em 20-04-2022. 22

Apêndice A

Features Extraídas do Dataset

Flow ID
Src IP
Src Port
Dst IP
Dst Port
Protocol
Timestamp
Flow Duration
Tot Fwd Pkts
Tot Bwd Pkts
TotLen Fwd Pkts
TotLen Bwd Pkts
Fwd Pkt Len Max
Fwd Pkt Len Min
Fwd Pkt Len Mean
Fwd Pkt Len Std
Bwd Pkt Len Max
Bwd Pkt Len Min
Bwd Pkt Len Mean
Bwd Pkt Len Std
Flow Byts/s
Flow Pkts/s
Flow IAT Mean
Flow IAT Std
Flow IAT Max
Flow IAT Min
Fwd IAT Tot
Fwd IAT Mean
Fwd IAT Std
Fwd IAT Max
Fwd IAT Min
Bwd IAT Tot
Bwd IAT Mean
Bwd IAT Std
Bwd IAT Max
Bwd IAT Min
Fwd PSH Flags
Bwd PSH Flags
Fwd URG Flags
Bwd URG Flags
Fwd Header Len
Bwd Header Len
Fwd Pkts/s
Bwd Pkts/s
Pkt Len Min
Pkt Len Max
Pkt Len Mean
Pkt Len Std
Pkt Len Var
FIN Flag Cnt
SYN Flag Cnt
RST Flag Cnt
PSH Flag Cnt
ACK Flag Cnt
URG Flag Cnt
CWE Flag Count
ECE Flag Cnt
Down/Up Ratio
Pkt Size Avg
Fwd Seg Size Avg
Bwd Seg Size Avg
Fwd Byts/b Avg
Fwd Pkts/b Avg
Fwd Blk Rate Avg
Bwd Byts/b Avg
Bwd Pkts/b Avg
Bwd Blk Rate Avg
Subflow Fwd Pkts
Subflow Fwd Byts
Subflow Bwd Pkts
Subflow Bwd Byts
Init Fwd Win Byts
Init Bwd Win Byts
Fwd Act Data Pkts
Fwd Seg Size Min

Active Mean
Active Std
Active Max
Active Min
Idle Mean
Idle Std
Idle Max
Idle Min
Label