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

# Um novo método baseado no modelo de Potts para detecção de intrusão em rede

Camila F. T. Pontes

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

Orientador
Prof. Dr. Marcelo Marotta

Coorientador
Prof. Dr. João Gondim

Brasília
2020

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

# Um novo método baseado no modelo de Potts para detecção de intrusão em rede

Camila F. T. Pontes

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

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

Prof. Dr. Luís Paulo Faina
CIC/UnB

Prof. Dr. Raul Matsushita
EST/UnB

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

Brasília, 10 de dezembro de 2020

# Dedicatória

Gostaria de dedicar este trabalho a todos que sofreram de alguma forma com a pandemia que enfrentamos em 2020. A todos que perderam familiares ou amigos. A todos que estão passando por dificuldades financeiras. A todos os profissionais de saúde. A todos que estão tendo dificuldade em manter a saúde mental. Não foi fácil para mim. Ainda assim, tem sido um tempo de muito crescimento pessoal. Enfim, pouco a pouco voltamos ao normal, ou ao "novo normal".

*A long long time ago*
*I can still remember how*
*That music used to make me smile*
*And I knew if I had my chance*
*That I could make those people dance*
*And maybe they'd be happy for a while*

*But February made me shiver*
*With every paper I'd deliver*
*Bad news on the doorstep*
*I couldn't take one more step*

Don McLean, "American Pie", 1971

# Agradecimentos

Agradeço aos meus orientadores, o professor Marcelo Marotta (CIC/UnB) e o professor João Gondim (CIC/UnB), por terem se dedicado junto comigo à elaboração deste trabalho. Aos professores Luís Paulo Faina (CIC/UnB) e Matt Bishop (UC Davis) por fazerem um "*sanity check*" da proposta inicial. À minha colega Manuela Souza por ter me ajudado na execução dos experimentos. Ao professor Werner Treptow (UnB/IB) e aos meus colegas do Labotatório de Biologia Teórica e Computacional (LBTC) pelos conhecimentos que obtive durante o período em que estive lá e pude aplicar neste trabalho. A todos os amigos que me apoiaram durantes os anos da graduação. À minha família, meus pais, meu irmão, meu companheiro, pelo incentivo e pelo carinho.

# Resumo

Sistemas de Detecção de Intrusão em Rede (NIDS, do inglês *Network Intrusion Detection Systems*) desempenham um importante papel como ferramentas para identificação de potenciais ameaças a redes de computadores. No contexto de crescentes volumes de tráfego de internet em redes de computadores, NIDS baseados em fluxos constituem boas soluções para o monitoramento de tráfego em tempo real. Nos últimos anos, diferentes classificadores de tráfego baseados em fluxos foram propostos utilizando aprendizagem de máquina. Entretanto, algoritmos de aprendizagem de máquina possuem algumas limitações. Além de requerer grandes quantidades de exemplos categorizados, que podem ser difíceis de obter, a maioria desses algoritmos não consegue se adaptar bem a diferentes domínios, *i.e.,* após serem treinados em um conjunto de dados específico, não são facilmente generalizáveis para outros conjuntos de dados. Por fim, muitos dos modelos inferidos por esses algoritmos são não interpretáveis. Para contornar essas limitações, é proposto um novo classificador de fluxos, chamado Energy-based Flow Classifier (EFC). EFC é um classificador baseado em anomalias que utiliza estatística inversa para inferir um modelo estatístico utilizando apenas exemplos benignos. É mostrado que o EFC é capaz de realizar classificação de fluxos de forma precisa e é mais adaptável a novos domínios do que algoritmos clássicos baseados em aprendizagem de máquina. Dados os bons resultados obtidos considerando três conjuntos de dados diferentes (CIDDS-001, CICIDS17 e CICDDoS19), o EFC se mostra como um algoritmo promissor para classificação robusta de fluxos de rede.

**Palavras-chave:** Detecção de intrusão em rede baseada em fluxo, detecção de intrusão em rede baseada em anomalia, classificação de fluxos de rede, sistemas de detecção de intrusão em rede, classificador de fluxos baseado em energias, modelo de Potts, adaptação a domínios

# Abstract

Network Intrusion Detection Systems (NIDS) play an important role as tools for identifying potential network threats. In the context of ever-increasing traffic volume on computer networks, flow-based NIDS arise as good solutions for real-time traffic classification. In recent years, different flow-based classifiers have been proposed using machine learning algorithms. Nevertheless, the classical machine learning algorithms have some limitations. For instance, they require large amounts of labeled data, which might be difficult to obtain. Additionally, most machine learning algorithms are not capable of domain adaptation, *i.e.,* after being trained on a specific dataset, they are not general enough to be applied to other related data distributions. And, finally, many of the models inferred by this algorithms are uninterpretable. To overcome these limitations, we propose a new flow-based classifier, called Energy-based Flow Classifier (EFC). This anomaly-based classifier uses inverse statistics to infer a statistical model based on labeled benign examples. We show that EFC is capable of accurately performing a one-class flow classification and is more adaptable to new domains than classical machine learning algorithms. Given the positive results obtained on three different datasets (CIDDS-001, CICIDS17 and CICDDoS19), we consider EFC to be a promising algorithm to perform robust flow-based traffic classification.

**Keywords:** Flow-based Network Intrusion Detection, Anomaly-based Network Intrusion Detection, Network Flow Classification, Network Intrusion Detection Systems, Energy-based Flow Classifier, Potts Model, Domain Adaptation

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Capítulo 1

# Introdução

O relatório de ameaças à segurança da internet da Symantec [2] aponta um crescimento de 56% no número de ataques *web* em 2019. Ataques de varredura de rede, de negação de serviço (DoS, do inglês *Denial of Service*) e de força bruta estão entre as ameaças mais comuns. Tais atividades maliciosas põe em risco não apenas indivíduos, mas também organizações coletivas como instituições de saúde pública, financeiras e governamentais. Nesse contexto, sistemas de detecção de intrusão (IDS, do inglês *Intrusion Detection Systems*) são importantes ferramentas para identificação de ameaças potenciais [3].

## 1.1 Histórico

Na década de 1970, a Força Aérea dos Estados Unidos (USAF, do inglês *United States Air Force*) se deparou com a necessidade de prover uso compartilhado de seus sistemas computacionais, os quais operavam em um ambiente altamente categorizado e possuíam uma base de usuários com vários níveis de credenciamento de segurança [4]. Nesse contexto, o civil James P. Anderson, financiado pela USAF, publicou, em 1980, um estudo intitulado *"Computer security threat monitoring and surveillance"* [5], no qual apresenta a ideia de uma detecção de intrusão automatizada.

Entre os anos de 1984 e 1986, Dorothy Denning e Peter Neumann desenvolveram o primeiro modelo de sistema de detecção de intrusão em tempo real. O protótipo recebeu o nome de *Intrusion Detection Expert System* (IDES) [6] e consistia em um sistema baseado em regras treinado para detectar atividade maliciosa [4]. Os trabalhos de Anderson e de Denning e Neumann despertaram o interesse da comunidade científica em relação à detecção de intrusão automatizada, de forma que as pesquisas nessa área avançaram rapidamente durante as décadas de 1980 e 1990 e continuam até os dias de hoje.

## 1.2 Sistemas de detecção de intrusão

A detecção de intrusão é uma abordagem utilizada na intenção de prover uma sensação de segurança para sistemas computacionais que operam em modo aberto [7]. A função de um IDS é monitorar os ativos de rede de forma a detectar uso malicioso ou comportamento anormal [8] (indicativos de uma possível tentativa de intrusão). De forma genérica, uma intrusão pode ser definida como qualquer conjunto de ações que comprometam a integridade, confidencialidade ou disponibilidade de um sistema [7].

Existem dois tipos de tecnologias clássicas de IDS: *host-based* IDS (HIDS) e *network-based* IDS (NIDS). *Network-based* IDS geralmente atuam como uma segunda linha de defesa, analisando o tráfego que passa pelo *firewall*. Já os *host-based* IDS atuam na última linha de defesa, analisando o tráfego de um usuário específico (Figura 1.1). Hoje em dia, existem ainda mais dois tipos de IDS: *wireless-based* IDS (WIDS) e *Network Behavior Analysis* (NBA). Os WIDS são similares aos NIDS, mas capturam tráfego de redes *wireless*. Os sistemas NBA inspecionam o tráfego de rede para reconhecer ataques com fluxos de tráfego inesperados [9]. A adoção de diferentes tecnologias de detecção de intrusão em um *mixed* IDS (MIDS) pode ser a melhor alternativa para uma detecção mais completa e acurada.[9].

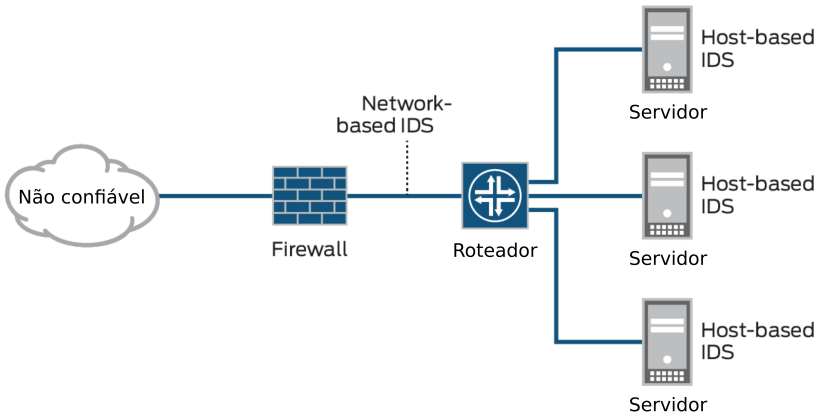


Figura 1.1: Desenho esquemático mostrando o posicionamento de dois tipos de IDS (*host-based* IDS e *network-based* IDS) em uma rede de computadores. *Network-based* IDS constituem uma segunda linha de defesa, posicionando-se atrás do *firewall*, enquanto *host-based* IDS fazem parte da última linha de defesa, instalados em máquinas específicas da rede. Figura adaptada de juniper.net (25/05/2018).

Algumas das vantagens dos *host-based* IDS é que eles analisam um volume menor de tráfego, são capazes de detectar ataques internos e são mais baratos e simples de implementar. Por outro lado, por terem um escopo mais específico, são incapazes de detectar ataques de rede (como *DNS spoofing*, *TCP hijacking*, *port scanning*, *ping of*

*death*, etc.) [10]. Já os *network-based* IDS, além de não consumirem recursos do usuário, são capazes de detectar ataques de rede e fazem uma detecção mais precoce da ameaça. Para uma comparação mais completa entre as diferentes tecnologias, ver trabalho de Liao e colaboradores, 2013 [9].

Os mecanismos de detecção de intrusão podem ser divididos três principais categorias: detecção baseada em assinatura (*signature/misuse-based*), detecção baseada em anomalia (*anomaly-based*) e *Stateful Protocol Analysis* (SPA) [9]. *Signature-based* IDS (Figura 1.2A) possuem um banco de dados de assinaturas de ataques conhecidos e tentam identificar essas assinaturas nos dados de rede analisados [11]. Já *anomaly-based* IDS (Figura 1.2B) constroem um modelo que descreve o tráfego normal e, então, marcam qualquer comportamento significativamente diferente do modelo como um ataque [11]. IDS que utilizam SPA funcionam de forma similar aos *anomaly-based* IDS, mas precisam conseguir determinar e rastrear os estados dos protocolos de rede, pois eles servem de base para as suas análises [9].

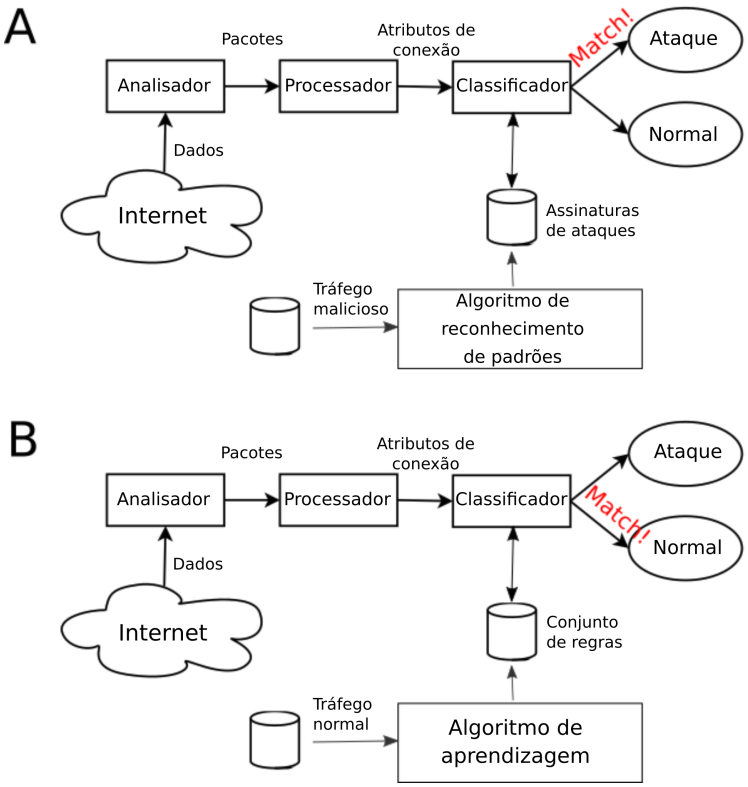


Figura 1.2: Arcabouço genérico de um *signature-based* IDS (A) e de um *anomaly-based* IDS (B). Figura adaptada de Esposito et al., *Intrusion Detection and Reaction* (*Intrusion Detection Systems* [1]), 2008.

Enquanto a detecção baseada em assinatura é favorecida em produtos comerciais por conta de sua previsibilidade e alta taxa de acurácia, os acadêmicos concebem a detecção baseada em anomalia como um método mais poderoso devido ao seu potencial teórico

para reconhecimento de novos ataques [12]. Outro motivo pelo qual *signature-based* IDS têm sido mais bem-sucedidos no mercado é o fato de serem mais simples de implementar. *Anomaly-based* IDS são difíceis de implementar e demandam uma infraestrutura de hardware específica.

## 1.3 Detecção de intrusão baseada em anomalia

Os *anomaly-based* IDS têm sido muito estudados pelos acadêmicos devido ao seu potencial de detecção de ataques *zero-day* [12]. Entretanto, a detecção de intrusão baseada em anomalia requer um bom conjunto de treino e uma definição cuidadosa do limiar de detecção, o que torna difícil a implementação desse tipo de sistema [11]. *Anomaly-based* IDS podem ser classificados em relação a:

- o tipo de algoritmo utilizado;
- a unidade de análise (pacotes individuais ou conjuntos de pacotes);
- o tipo de dado analisado (cabeçalho ou *payload*).

Em relação ao tipo de algoritmo utilizado, pode-se definir duas principais categorias: algoritmos baseados em modelos estatísticos e algoritmos baseados em redes neurais. Os primeiros são dominantes, sendo utilizado por mais de 50% dos sistemas existentes [11]. Nesses sistemas, o algoritmo (durante a chamada fase de treino) constrói um modelo estatístico do comportamento legítimo da rede; em seguida, na fase de detecção, os dados de entrada são comparados com o modelo por meio de uma função de distância e, quando a distância medida supera um determinado limiar, a entrada é considerada anômala (um ataque) [11]. *Anomaly-based* IDS baseados em redes neurais funcionam de forma similar, mas, ao invés de construir um modelo estatístico, eles treinam uma rede neural para reconhecer o comportamento do tráfego legítimo [11].

Quanto à unidade analisada, existem sistemas orientados a pacote e sistemas orientados a fluxos. Um sistema orientado a pacotes utiliza os pacotes como unidade mínima de informação, enquanto um sistema orientado a fluxos considera as características de toda a comunicação [11]. Na prática, sistemas orientados a fluxos levam em conta o número de bytes enviados/recebidos, a duração da conexão e o protocolo de camada de transporte utilizado. A maioria dos *anomaly-based* IDS são orientados a pacote [11].

Em relação ao tipo de dado analisado, existem os sistemas baseados em cabeçalho e os sistemas baseados em *payload*. Alguns exemplos de ataques detectáveis por análise de cabeçalho são o *Teardrop Exploit* (DoS) e o *Land Attack* (DoS), que exploram vulnerabilidades na camada de transporte. Alguns exemplos de ataques detectáveis por análise de

*payload* são *SQL Injection* e *PHF Attack*, que exploram vulnerabilidades na camada de aplicação [11].

Um IDS é tão eficiente quanto os aspectos relevantes que ele consegue extrair do comportamento do usuário na etapa de construção de modelos [13]. A etapa de treino é essencial para o bom-funcionamento de IDS e, ainda hoje, é um desafio construir bons modelos a partir de fontes de informação muito ruidosas. Entretanto, à medida em que os algoritmos de inteligência artificial (AI, do inglês *artificial intelligence*) se aproximam dos mecanismos de percepção do cérebro humano, aprendendo a criar abstrações e a ignorar elementos não relevantes, os IDS se tornam mais robustos e cada vez mais adequados para serem aplicados em sistemas reais.

## 1.4 Contribuições

Conforme mencionado na seção anterior, existem duas abordagens principais para *Network Intrusion Detection System*s (NIDSs) em relação do tipo de dado analisado: baseada em pacotes e baseada em fluxos. Na primeira, uma inspeção profunda de pacotes é realizada levando em conta o *payload* de cada pacote individualmente, bem como as informações do cabeçalho [14]. Na segunda, fluxos, *i.e.,* conjuntos de pacotes, são analisados levando-se em conta algumas propriedades, *e.g.,* duração, número de pacotes, número de *bytes* e porta de origem/destino [14]. Para que uma classificação de tráfego em tempo real seja possível, um grande número de pacotes precisam analisados em um curto intervalo de tempo, tornando a primeira abordagem custosa demais para ser aplicada. Por outro lado, uma vez que abordagens baseadas em fluxo permitem a classificação de todo o tráfego através da inspeção do equivalente a 0.1% do volume total, NIDSs baseados na análise de fluxos surgem como boas soluções para classificação em tempo real [15].

Ao longo dos últimos anos, diversos classificadores de tráfego baseados em fluxos utilizando *Machine Learning* (ML) foram propostos [16]. De acordo com Umer et al. [16], os melhores classificadores apresentam um ótimo desempenho, com aproximadamente 99% de precisão. Apesar da boa precisão, classificadores baseados em ML precisam de grandes quantidades de amostras de tráfego rotuladas para treino. A rotulagem de amostras de tráfego real, entretanto, pode ser difícil, especialmente no caso de tráfego malicioso [17]. Além disso, classificadores baseados em ML funcionam melhor em contextos estáticos, *i.e.,* após serem treinados com base em uma distribuição de dados específica, geralmente não se adaptam a *concept drift* [18] ou mudanças de domínio [19, 20, 21]. Por fim, a maioria dos algoritmos de ML gera modelos não interpretáveis, difíceis de serem analisados e reajustados, caso necessário [22, 23]. Nesse sentido, há a clara necessidade de um novo classificador baseado em fluxos para NIDSs capaz de inferir um modelo interpretável,

com treinamento baseado apenas em exemplos benignos e que seja adaptável a diferentes domínios.

Neste trabalho, é proposto um novo classificador de fluxos baseado na estatística inversa do modelo de Potts, chamado *Energy-based Flow Classifier* (EFC). O EFC é um algoritmo interpretável que realiza classificação unária (*one-class classification*), *i.e.,* um modelo estatístico é inferido a partir de exemplos de apenas uma classe (fluxos benignos) e posteriormente utilizado para diferenciar duas classes (fluxos benignos e fluxos maliciosos). O desempenho do EFC foi comparado ao de uma variadade de classificadores considerando três *datasets* diferentes: CIDDS-001 [24], CICIDS17 [25] e CICDDoS19 [26]. Os resultados obtidos mostram que o desempenho do EFC sobre os *datasets* considerados é comparável ao desempenho dos demais classificadores. Além disso, foi constatado que o EFC é menos sensível a mudanças na distribuição de dados (mudança de domínio) do que os outros classificadores. As principais contribuições deste trabalho são:

- proposta e implementação de um novo classificador de fluxos para NIDSs baseado na estatística inversa do modelo de Potts, o EFC;
- comparação do desempenho do EFC com classificadores baseados em ML considerando três *datasets* diferentes para teste;
- análise do desempenho de diferentes classificadores quando treinados em um domínio e testados em outro domínio relacionado.

O restante deste trabalho é estruturado da seguinte maneira. No Capítulo 2, é feita uma breve descrição do estado da arte em NIDSs baseados em fluxos. No Capítulo 3, são apresentados alguns conceitos iniciais, bem como uma descrição dos *datasets* utilizados. Além disso, são apresentadas as bases teóricas do EFC e sua implementação. No Capítulo 4, são apresentados os resultados obtidos em relação à aplicabilidade do EFC e ao seu desempenho comparativo. Finalmente, no Capítulo 5, são apresentadas as conclusões e perspectivas futuras.

# Capítulo 2

# Trabalhos relacionados

Neste capítulo, inicialmente é feita uma breve descrição do estado da arte em detecção de intrusão baseada em fluxos de rede, com a apresentação de trabalhos recentes. Além disso, são apresentados alguns trabalhos que utilizam os *datasets* CIDDS-001, CICIDS17 e CICDDoS19. Por fim, são abordados os desafios associados ao uso de aprendizagem de máquina mencionados na introdução, *i.e.,* a dificuldade de obtenção de amostras rotuladas, a dificuldade de adaptação a novas distribuições após a etapa de treino e a geração de modelos não interpretáveis.

## 2.1 Classificação de fluxos de rede utilizando aprendizagem de máquina

Diversos classificadores de fluxos baseados em ML foram explorados ao longo dos últimos anos para detecção de intrusão em redes de computadores. Em [16] é feita uma extensa revisão de literatura sobre classificação de tráfego de rede baseada em fluxos. Já em [27, 28, 29], são realizadas comparações do desempenho de diferentes classificadores de tráfego baseados em ML. Por fim, [30, 31, 32, 33] ilustram como NIDS tem sido aplicados em contextos modernos de redes *Internet of Things* (IoT) e ambientes de computação em nuvem. Esses estudos serão detalhados nos parágrafos seguintes.

Umer et al. publicaram em 2017 uma vasta revisão de literatura sobre detecção de intrusão baseada em fluxo [16]. Nesse trabalho, são mencionadas algumas desvantagens de se utilizar métodos baseados em ML para classificação de tráfego, como, por exemplo, o alto custo computacional da etapa de treino, a dificuldade em obter conjuntos de treino representativos e a alta taxa de falsos positivos. O presente trabalho contribui no sentido de solucionar alguns desses problemas, uma vez que o algoritmo proposto aqui tem baixo custo computacional e aprende exclusivamente com tráfego benigno.

Vinayakumar et al. (2017) [27] utilizaram os *datasets* KDDCup'99 e NSL-KDD para avaliar o desempenho de diferentes classificadores baseados em ML, com o objetivo específico de comparar *shallow-* e *deep-learning*. Os classificadores considerados são baseados em *Linear Regression* (LR), *Naive Bayes* (NB), *K-Nearest Neighbors* (KNN), *Decision Tree* (DT), Adaboost, *Random Forest* (RF), *Support Vector Machine* (SVM) (*shallow-learning*), bem como *Multi-Layer Perceptron* (MLP) e *Deep Belief Network* (DBN) (*deep-learning*). As métricas utilizadas para avaliação do desempenho foram a acurácia e o F1-score. A conclusão do trabalho foi que, na maioria dos cenários testados, *deep-learning* apresentou melhor desempenho do que *shallow-learning*. No presente trabalho, são utilizados tanto classificadores baseados em *shallow-* quanto em *deep-learning* como parâmetros de comparação para o EFC. Aqui, entretanto, ao invés de realizarmos uma classificação multi-classe, a classificação realizada é binária.

Mais recentemente, Mahfouz et al. (2020) [28] também realizaram uma avaliação do desempenho de diferentes classificadores baseados em ML utilizando o *dataset* NSL-KDD, que contém as seguintes classes de ataques: *DoS*, *Probe*, *R2L* e *U2L*, além de tráfego benigno. Os classificadores avaliados foram construídos utilizando NB, *Artificial Neural Network* (ANN), SVM, KNN e DT e as métricas utilizadas para comparação foram a acurácia, o F1-score e a área sobre a curva ROC. O classificador que obteve melhor desempenho sem seleção de atributos foi o baseado em DT. Já com seleção de atributos, o classificador baseado em KNN foi o melhor. No presente trabalho, também vamos avaliar o desempenho do EFC em comparação ao desempenho de classificadores baseados em NB, ANN, SVM, KNN, DT, além de RF, que está faltando no trabalho de Mahfouz et al..

Por fim, Khan et al. (2020) [29] também fizeram uma comparação do desempenho de diferentes classificadores baseados em ML. Os classificadores avaliados são baseado em DT, RF, *Gradient Boosting* (GB), AdaBoost e NB e o dataset utilizado foi o UNSW-NB15. Com base nas métricas acurácia, F1-score e taxa de falsos positivos, o classificador baseado em RF foi que obteve o melhor desempenho. De fato, o RF tem sido aplicado em diversos NIDSs recentes [34, 35, 36]. No presente trabalho, vamos realizar a comparação do desempenho do EFC com classificadores baseados em RF e outros algoritmos de ML em um experimento de classificação binária. Assim como nesse e nos outros trabalhos mencionados, também utilizaremos o F1-score como métrica de avaliação de desempenho.

A detecção de intrusão baseada em fluxos de rede também tem sido explorada em contextos modernos, como em redes IoT [30, 31] e em ambientes de computação em nuvem [33, 32]. As soluções propostas para detecção de intrusão em ambientes IoT e de computação em nuvem atingiram patamares satisfatórios de precisão, com baixos tempos de execução nos testes realizados. Entretanto, sua capacidade de adaptação a diferentes

domínios ainda precisa ser investigada. Neste trabalho, a capacidade de adaptação a domínios de diversos classificadores é investigada. Na próxima seção, são descritos alguns trabalhos recentes que justificam a escolha dos *datasets* utilizados no presente trabalho.

## 2.2 Classificação de fluxos de rede utilizando CIDDS-001, CICIDS17 e CICDDoS19

Para avaliar o desempenho do EFC, um dos *datasets* utilizados é o CIDDS-001. Esse *dataset* foi utilizado por Verma e Ranga (2018) [37] para avaliar o desempenho dos algoritmos KNN e *k-means clustering* para classificação de tráfego de rede. Ambos os algoritmos atingiram valores de acurácia superiores a 99%. Além disso, Ring *et al.* [38] investigaram a detecção de varreduras de porta utilizando o CIDDS-001. A abordagem proposta por eles foi capaz de reconhecer precisamente os ataques com uma baixa taxa de falsos alarmes. Por fim, Abdulhammed *et al.* [39] também investigaram classificação baseada em fluxos utilizando o CIDDS-001 e propuseram uma abordagem que foi considerada robusta para tráfego de rede desbalanceado. O CIDDS-001 contém amostras de fluxos benignos, bem como os mais recentes ataques cibernéticos e possui cenários de tráfego simulado e de tráfego real, sendo um *dataset* relevante a ser considerado em estudos sobre classificação de tráfego baseada em fluxos.

Outros dois *datasets* utilizados neste trabalho são o CICIDS17 e o CICDDoS19, do Instituto Canadense de Segurança Cibernética. O CICIDS17 foi recentemente utilizado por Yulianto, Sukarno e Suwastika para avaliar o desempenho de um classificador baseado em Adaboost [40]. Aksu e colaboradores fizeram o mesmo em 2018 com diferentes classificadores baseados em ML [41]. O CICIDS17 é um *dataset* atual e relevante para avaliação de soluções para classificação de fluxos de rede, sendo, portanto, mais um dos *datasets* escolhidos para avaliar o desempenho do EFC.

Por fim, o CICDDoS19 é um *dataset* bastante atual com foco em ataques de negação de serviço distribuídos (*Distributed Denial of Service*, DDoS). Um trabalho recente [42] propões um NIDS de tempo real baseado em entropia para detecção de DDoS volumétricos em IoT e realiza testes sobre o *dataset* CICDDoS19, dentre outros. Outro trabalho recente [43] obteve 99% de acurácia de classificação sobre o *dataset* CICDDoS19 utilizando uma *Convolutional Neural Network* (CNN). E, por fim, Novaes *et al.* [44] propõe um sistema para detecção de intrusão baseado em lógica fuzzy, o qual também foi testado sobre o CICDDoS19. Isso mostra que esse *dataset* é relevante e tem sido utilizado para testar diferentes NIDSs. Neste trabalho, o CICDDoS19 e outros dois *datasets* atuais são utilizados para testar o EFC e comparar o seu desempenho na classificação de tráfego ao desempenho de outros classificadores baseado em ML.

## 2.3 Desafios: falta de amostras rotuladas, adaptabilidade e modelos não interpretáveis

Um dos problemas comumente citados na área de aprendizagem de máquina é a dependência que a maioria dos algoritmos possuem de uma grande quantidade de amostras rotuladas para treino [17], as quais podem, muitas vezes, ser difíceis de se obter. Em especial, amostras de tráfego malicioso são difíceis de se obter no mundo real e, portanto, quase todos os *datasets* de segurança de redes incluem somente ataques simulados. Um dos problemas que isso pode causar é a dificuldade de detecção de ataques *zero-day* [17], que, por definição, não estão presentes em nenhum conjunto de treino. Diante disso, em princípio, a única estratégia possível para detecção de ataques *zero-day* seria utilizando uma classificador baseado em anomalia [45], como o classificador proposto neste trabalho. Portanto, uma vez que o EFC não utiliza amostras maliciosas para treino, o problema da dificuldade de se obter amostras maliciosas rotuladas é contornado, bem como o problema de desbalanceamento presente na maioria dos *datasets*.

Outro problema comum em ML é a baixa capacidade de predição dos modelos em diferentes domínios (ou distribuições de dados) após a fase de aprendizagem. De fato, a maioria dos algoritmos de ML perde parte do seu poder preditivo quando exposto a distribuições de dados um pouco diferentes da distribuição utilizada para treino [19]. Na área de segurança de redes, essa adaptabilidade é especialmente importante devido à existência de ataques *zero-day* e à artificialidade dos *datasets* de tráfego de rede utilizados na academia. Zolanvari et al. (2018) [19] fazem uma discussão muito interessante sobre as diferenças existentes entre os *datasets* utilizados na academia para teste de NIDS e o tráfego de rede observado no mundo real. Complementarmente, os trabalhos de Bartos *et al.* (2016) [20] e Li *et al.* (2019) [21] também abordam a temática da adaptação a novos domínios. Eles propõe abordagens similares entre si para tratar essa questão, aplicando transformações aos dados para reduzir as diferenças na distribuição dos atributos entre domínios. No presente trabalho, é proposto um classificador que é intrinsecamente adaptável a diferentes domínios,uma vez que sua aprendizagem é baseada apenas em exemplos benignos.

Por fim, outra grande questão é a geração de modelos não interpretáveis [23]. ANNs, em especial, têm se tornado cada vez mais opacas com o passar do tempo, apesar de apresentarem desempenho superior a outros algoritmos em diversas tarefas. Holzinger et al. (2018) [23] ressaltam que os melhores algoritmos de aprendizagem de máquina não são compreensíveis e que, portanto, as decisões tomadas com base neles não podem ser explicadas. Entretanto, cada vez mais, diferentes contextos exigem soluções transparentes e, por isso, a explicabilidade do modelo é tão importante. O EFC, por ser um classificados

que produz um modelo transparente, se enquadra bem nesse contexto moderno, o qual exige que as decisões tomadas possam ser analisadas em retrospectiva. Portanto, no próximo capítulo, é descrito o EFC, uma nova proposta de classificador interpretável, que treina apenas com amostras benignas e é mais resiliente a mudanças de domínio.

# Capítulo 3

# Proposta

Neste capítulo, a abordagem teórica proposta para classificação binária de fluxos de rede será introduzida. Essa abordagem teórica foi adaptada da física estatística, de um modelo teórico da interação de *spins* em malha cristalina. Utilizando essa teoria, foi construído um classificador de fluxos baseado em anomalias, *i.e.,* um algoritmo que aprende características de fluxos benignos (infere um modelo estatístico representativo dessa classe) na etapa de treino e utiliza as características aprendidas para classificar fluxos em benignos ou maliciosos na etapa de teste.

## 3.1 Descrição do modelo estatístico

O principal objetivo da estatística inversa é a inferência de uma distribuição estatística global baseada em uma pequena amostra dessa distribuição [46]. Os métodos que utilizam estatística inversa obtiveram grande sucesso em problemas de outras áreas do conhecimento, *e.g.,* o problema da predição de contatos entre proteínas na Biofísica [46, 47]. Neste trabalho, a inferência estatística é baseada no modelo de Potts [48]. Esse modelo faz uma descrição matemática de um conjunto de spins interagindo em uma malha cristalina. Nesse modelo, o conjunto de spins que interagem é mapeado em um grafo $G(\eta, \varepsilon)$ (ver Figura 3.1 A)), onde cada nó $i \in \eta = \{1, ..., N\}$ possui um spin associado $a_i$, que pode assumir valores de um conjunto $\Omega$ que contém todos os possíveis estatos quânticos para um spin isolado. Cada nó $i$ possui também um campo local $h_i(a_i)$ que é função do estado de $a_i$. Além disso, cada aresta $(i, j) \in \varepsilon$, $i, j \in \eta$, possui um valor de acoplamento $e_{ij}(a_i, a_j)$ associado, que é função dos estados dos spins $a_i$ and $a_j$ associados aos nós $i$ and $j$. Cada configuração específica do sistema de spins possui uma energia total associada, determinada pelo Hamiltoniano $\mathcal{H}(a_1...a_N)$, que depende do estado de todos os spins.

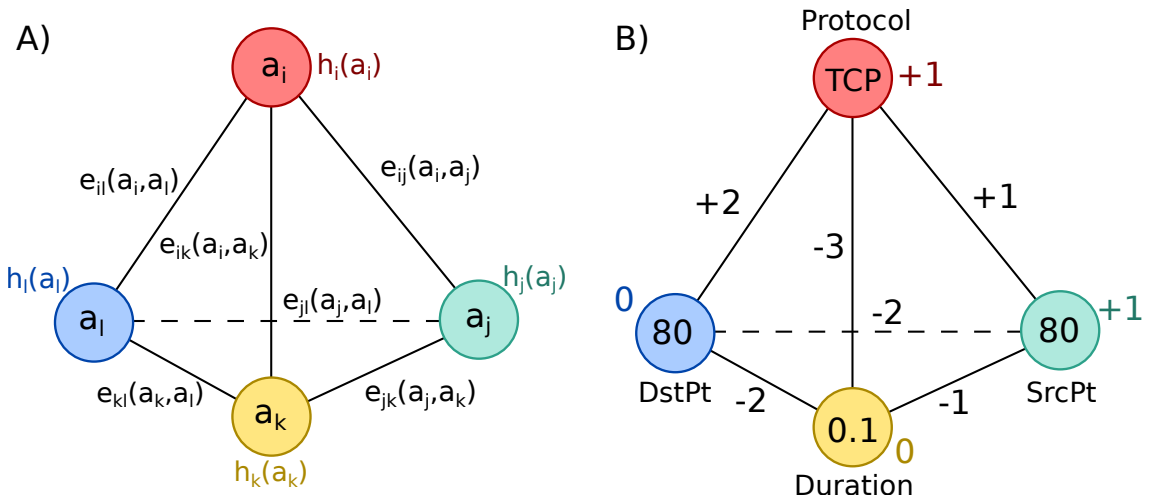


Figura 3.1: A) Modelo de Potts para spins que interagem em uma malha critalina. B) Fluxo de rede mapeado em um grafo.

Neste trabalho, o modelo de Potts é adaptado para representar fluxos de rede (ver Figura 3.1 B)). Um fluxo individual $k$ é representado por uma configuração específica do grafo $G_k(\eta, \varepsilon)$. Ao invés de spins, cada nó representa um atributo $i \in \eta = \{SrcPort, ..., Flags\}$. Em cada fluxo $k$, cada atributo $i$ assume um valor $a_{ki}$ do conjunto $\Omega_i$, que contém todos os possíveis valores para aquele atribulo específico. Da mesma forma que no modelo de Potts, cada atributo $i$ possui um campo local $h_i(a_{ki})$ associado. Além disso, $\varepsilon = \{(i,j)|i,j \in \eta; i \neq j\}$ representa o conjunto de arestas composto por todos os possíveis pares de atributos. Cada aresta possui um valor de acoplamento associado, determinado pela função $e_{ij}(a_{ki}, a_{kj})$.

Uma vez que os valores dos campos locais e dos acoplamentos dependem dos valores assumidos pelos atributos em um dado fluxo, cada fluxo possui uma combinação específica dessas quantidades. Da mesma forma que no modelo de Potts, campos locais e acoplamentos determinam a "energia"total $\mathcal{H}(a_{k1}...a_{kN})$ de cada fluxo. Por exemplo, na Figura 3.1 B), a "energia"total do fluxo é obtida a partir da soma de todos os valores associados às arestas e aos nós, resultando em um total de -3. Note que o conceito de energia aqui é análogo à noção de Hamiltoniano na Mecânica Quântica. É importante ressaltar que o modelo descrito aqui é discreto e, portanto, atributos contínuos devem ser discretizados para se adaptar ao modelo. As classes para discretização de atributos são mostradas nos apêndices. A seguir, será apresentado o arcabouço teórico aplicado para realizar a inferência do modelo estatístico e subsequente classificação de fluxos baseada nos valores de energia.

## 3.2 Inferência do modelo estatístico

Nesta seção, um modelo estatístico será inferido em termos dos valores de acoplamentos e campos locais para, posteriormente, realizar uma classificação baseada em energias. A ideia principal é obter um modelo estatístico a partir de amostras de fluxos benignos e inferir valores de acoplamentos e campos locais que caracterizam esse tipo de tráfego. Quando forem calculadas as energias de fluxos benignos e maliciosos não rotulados utilizando os parâmatros inferidos, é esperado que os fluxos benignos tenham menores valores de energias do que os fluxos maliciosos.

Seja $(A_1...A_{\mathrm{N}})$ uma N-tupla de atributos, que pode ser instanciada para o fluxo $k$ como $(a_{k1}...a_{k\mathrm{N}})$, with $a_{k1} \in \Omega_1, ..., a_{k\mathrm{N}} \in \Omega_{\mathrm{N}}$. O valor de cada atributo $a_{ki}$ é mapeado em um inteiro do conjunto $\Omega = \{1, 2, ..., Q\}$, *i.e.,* os alfabetos de todos os atributos são o mesmo $\Omega_i = \Omega$, de tamanho $Q$. Se um dado atributo pode assumir apenas $M$ valores e $M < Q$, é considerado que os valores $M + 1, ..., Q$ apresentam probabilidade nula. Por exemplo, se os únicos valores possíveis para o atributo *protocol* são { *'TCP'* e *'UDP'*}, e dado $Q = 4$. Nesse caso, teríamos o mapa { *'TCP'*:1, *'UDP'*:2, *' '*:3, *' '*:4 } e os valores 3 e 4 nunca ocorreriam na prática.

Seja $\mathcal{K}$ o conjunto de todos os fluxos possíveis, *i.e.,* todas as possíveis combinações de valores de atributos ($\mathcal{K} = \Omega^N$), e seja $\mathcal{S} \subset \mathcal{K}$ um conjunto de fluxos. É possível utilizar estatística inversa para inferir um modelo estatístico que associa uma probabilidade $P(a_{k1}...a_{k\mathrm{N}})$ a cada fluxo $k \in \mathcal{K}$, com base no conjunto $\mathcal{S}$. O modelo estatístico global $P$ é inferido segundo o Princípio da Maximização da Entropia [49]:

$$\max_{P} \quad -\sum_{k \in \mathcal{K}} P(a_{k1}...a_{k\mathrm{N}}) log(P(a_{k1}...a_{k\mathrm{N}})) \tag{3.1}$$

$$s.t.$$

$$\sum_{k \in \mathcal{K} | a_{ki} = a_i} P(a_{k1}...a_{k\mathrm{N}}) = f_i(a_i) \tag{3.2}$$

$$\forall i \in \eta;\ \forall a_i \in \Omega;$$

$$\sum_{k \in \mathcal{K} | a_{ki} = a_i, a_{kj} = a_j} P(a_{k1}...a_{k\mathrm{N}}) = f_{ij}(a_i, a_j) \tag{3.3}$$

$$\forall (i, j) \in \eta^2 \mid i \neq j;\ \forall (a_i, a_j) \in \Omega^2;$$

onde $f_i(a_i)$ é a frequência empírica do valor $a_i$ no atributo $i$ e $f_{ij}(a_i, a_j)$ é a frequência empírica conjunta do par de valores $(a_i, a_j)$ nos atributos $i$ e $j$. Note que as restrições 3.2 e 3.3 forçam o modelo $P$ a gerar as frequências empíricas simples e duplas como marginais. Isso é uma forma de garantir que o modelo reproduza os dados empíricos.

As frequências empíricas simples e duplas, $f_i(a_i)$ e $f_{ij}(a_i, a_j)$, são obtidas a partir do conjunto $\mathcal{S}$, fazendo a contagem das ocorrências de um dado valor de atributo $a_i$ ou de pares de valores de atributos $(a_i, a_j)$, respectivamente, e dividindo pelo número total de fluxos em $\mathcal{S}$. Uma vez que o conjunto $\mathcal{S}$ é finito e muito menor que $\mathcal{K}$, inferências baseadas em $\mathcal{S}$ estão sujeitas a efeitos de subamostragem (*undersampling*). Seguindo a abordagem proposta em [47], foi utilizada pseudocontagem na estimativa das frequências empíricas para limitar os esfeitos negativos da subamostragem, realizando as seguintes correções:

$$f_i(a_i) \leftarrow (1-\alpha) f_i(a_i) + \frac{\alpha}{Q} \tag{3.4}$$

$$f_{ij}(a_i, a_j) \leftarrow (1-\alpha) f_{ij}(a_i, a_j) + \frac{\alpha}{Q^2} \tag{3.5}$$

onde $(a_i, a_j) \in \Omega^2$ and $0 \leq \alpha \leq 1$ é uma parâmetro que define o peso relativo das pseudocontagens. A introdução de pseudocontagens é equivalente a assumir que $\mathcal{S}$ seja extendido para conter uma certa fração extra de fluxos com valores de atributos uniformemente distribuídos.

A maximização proposta pode ser resolvida utilizando um Lagrangiano, conforme descrito por Jaynes [49], resultando na seguinte distribuição Boltzmann-like:

$$P^*(a_{k1}...a_{k\mathrm{N}}) = e^{-\mathcal{H}(a_{k1}...a_{k\mathrm{N}})} \tag{3.6}$$

onde

$$\mathcal{H}(a_{k1}...a_{k\mathrm{N}}) = -\sum_{i,j|i<j} e_{ij}(a_{ki}, a_{kj}) - \sum_i h_i(a_{ki}) \tag{3.7}$$

é o Hamiltoniano do fluxo $k$. O objetivo aqui é calcular as energias de fluxos individuais, *i.e.,* o Hamiltoniano de cada fluxo, conforme descrito na eq. (3.7).

Note que o Hamiltoniano, conforme apresentado aqui, é completamente determinado em termos dos multiplicadores de Lagrange $e_{ij}(\cdot)$ e $h_i(\cdot)$ associados às restrições (3.2) e (3.3), respectivamente. No arcabouço teórico do modelo de Potts, os multiplicadores de Lagrange têm um significado especial, sendo $\{e_{ij}(a_i, a_j) | (a_i, a_j) \in \Omega^2\}$ o conjunto de todos os possíveis valores de acoplamentos entre dois atributos $i$ e $j$, e $\{h_i(a_i) | a_i \in \Omega\}$ o conjunto de todos os possíveis campos locais associados a um atributo $i$.

A inferência dos campos locais e dos acoplamentos par-a-par é difícil, pois o número de parâmetros excede o número de restrições independentes. Felizmente, por conta das propriedades físicas de spins que interagem, é possível inferir os valores dos acoplamentos $e_{ij}(a_i, a_j)$ utilizando uma aproximação Gaussiana. Assumindo que essa aproximação também pode ser feita no caso de atributos de fluxos que interagem, os acoplamentos são

inferidos da seguinte maneira:

$$e_{ij}(a_i, a_j) = -(C^{-1})_{ij}(a_i, a_j), \tag{3.8}$$
$$\forall (i,j) \in \eta^2, \forall (a_i, a_j) \in \Omega^2, a_i, a_j \neq Q$$

onde

$$C_{ij}(a_i, a_j) = f_{ij}(a_i, a_j) - f_i(a_i) f_j(a_j) \tag{3.9}$$

é a matriz de covariância obtida a partir das frequências empíricas simples e conjuntas. A inversão da matriz de covariância é uma técnica estatística comumente utilizada para remoção dos efeitos de correlações indiretas nos dados [50]. É importante ressaltar que o número de restrições independentes nas eq. (3.2) e eq. (3.3) é, na verdade, $\frac{N(N-1)}{2}(Q-1)^2 + N(Q-1)$, ainda que o modelo na eq. (3.6) tenha $\frac{N(N-1)}{2}Q^2 + NQ$ parâmetros. Portanto, sem perda de generalidade, definimos:

$$e_{i,j}(a_i, Q) = e_{i,j}(Q, a_j) = h_i(Q) = 0 \tag{3.10}$$

Portanto, na eq. (3.8) não há necessidade de calcular $e_{i,j}(a_i, a_j)$ caso $a_i$ ou $a_j$ seja igual a $Q$ [47]. Em seguida, os campos locais $h_i(a_i)$ podem ser inferidos utilizando uma aproximação de campo médio [51]:

$$\frac{f_i(a_i)}{f_i(Q)} = exp\left(h_i(a_i) + \sum_{j,a_j} e_{ij}(a_i, a_j) f_j(a_j)\right), \tag{3.11}$$
$$\forall i \in \eta, a_i \in \Omega, a_i \neq Q$$

onde $f_i(Q)$ é a frequência do último elemento $a_i = Q$ para qualquer atributo $i$, utilizada para normalização. Também é importante mencionar que o elemento $Q$ foi selecionado arbitrariamente e poderia ser substituído por qualquer outro valor em $\{1, \ldots, Q\}$ desde que o elemento selecionado fosse mantido o mesmo no cálculo dos campos locais para todos os atributos $i \in \eta$. Note que na eq. (3.11), as frequências simples empíricas $f_i(a_i)$ e os valores dos acoplamentos $e_{ij}(a_i, a_j)$ são conhecidos, resultando em:

$$h_i(a_i) = ln\left(\frac{f_i(a_i)}{f_i(Q)}\right) - \sum_{j,a_j} e_{ij}(a_i, a_j) f_j(a_j) \tag{3.12}$$

Na aproximação de campo médio apresentada acima, a interação de um dado atributo com os seus vizinhos é substituída pela interação aproximada com um atributo médio, *i.e.,* a média dos atributos, resultando em um valor aproximado para o campo local associado a ele.

Para maiores detalhes sobre os cálculos apresentados, veja [46]. Uma vez que os parâmetros do modelo são conhecidos, é possível calcular a energia de um fluxo qualquer, conforme apresentado na eq. (3.7). A seguir, será apresentada a implementação do arcabouço teórico utilizado para realizar a classificação de fluxos de rede em benignos ou maliciosos.

## 3.3 Classificação baseada em energias

A energia de um dado fluxo pode ser calculada conforme a eq. (3.7), tendo como base os valores específicos dos atributos no fluxo e os paraâmetros do modelo estatístico inferidos na seção anterior. De forma simplificada, a energia de um dado fluxo é a soma negativa dos acoplamentos e dos campos locais associados aos seus atributos, conforme um dado modelo estatístico. Isso significa que, se um fluxo possui atributos comuns no conjunto de fluxos utilizado para inferir o modelo, esse fluxo terá baixa energia. Isso ocorre porque os atributos que aparecem fortemente acoplados no conjunto $\mathcal{S}$ que gerou aquele modelo estavam presentes no fluxo considerado.

Uma vez que o EFC é um classificador baseado em anomalias, o modelo estatístico utilizado para classificação é inferido com base em amostras de fluxos benignas. É de se esperar, portanto, que na etapa de teste energias de amostras benignas sejam menores do que energias de amostras maliciosas. Nesse sentido, é possível classificar amostras de fluxos como benignas ou maliciosas com base em um dado limiar de energia. A classificação é realizada partindo do princípio de que as amostras com energia abaixo do limiar são benignas e amostras com energia maior ou igual ao limiar são maliciosas. Note que o limiar para classificação pode ser definido de diferentes maneiras, podendo ser tanto estático quanto dinâmico. Neste trabalho, será considerado um limiar estático.

O Algoritmo 1 mostra a implementação do EFC. Nas linhas 2-5, o modelo estatístico é inferido, conforme descritos nas eqs. (3.4), (3.5), (3.8) e (3.12). Em seguida, nas linhas 6-27, o NIDS monitora a rede esperando que um fluxo seja capturado. Quando um fluxo é capturado, sua energia é calculada nas linhas 9-20, segundo o Hamiltoniano apresentado na eq. (3.7). A energia calculada é, então, comparada ao valor de um limiar predeterminado (*cutoff*) na linha 21. Caso a energia seja maior ou igual ao limiar, o fluxo é classificado como suspeito e deve ser encaminhado para inspeção de pacotes (linha 23). Caso contrário, o fluxo é liberado e o NIDS retorna ao estado de espera até que um novo fluxo seja capturado.

É importante ressaltar que a complexidade temporal da etapa de treino do EFC é $O((M \times Q)^3 + N \times M^2 \times Q^2)$, onde $N$ é o número de amostras de fluxos, $M$ é o número de atributos de cada fluxo e $Q$ é o tamanho do alfabeto. Por sua vez, a complexidade da

**Algorithm 1** Energy-based Flow Classifier

**Input:** $normal\_flows_{(K \times N)}$, $Q$, $\alpha$, $cutoff$

import all model inference functions
$f\_i \leftarrow SiteFreq(normal\_flows, Q, \alpha)$
$f\_ij \leftarrow PairFreq(normal\_flows, f\_i, Q, \alpha)$
$e\_ij \leftarrow Couplings(f\_i, f\_ij, Q)$
$h\_i \leftarrow LocalFields(e\_ij, f\_i, Q)$
**while** Scanning the Network **do**
    $flow \leftarrow$ wait_for_incoming_flow()
    e $\leftarrow 0$
    **for** $i \leftarrow 1$ to $N - 1$ **do**
        $a\_i \leftarrow flow[i]$
        **for** $j \leftarrow i + 1$ to $N$ **do**
            $a\_j \leftarrow flow[j]$
            **if** $a\_i \neq Q$ and $a\_j \neq Q$ **then**
                $e \leftarrow e - e\_ij[i, a\_i, j, a\_j]$
            **end if**
        **end for**
        **if** $a\_i \neq Q$ **then**
            $e \leftarrow e - h\_i[i, a\_i]$
        **end if**
    **end for**
    **if** $e \geq cutoff$ **then**
        stop_flow()
        forward_to_DPI()
    **else**
        release_flow()
    **end if**
**end while**

etapa de classificação para cada amostra é $O(M^2)$. Isso significa que, em ambas etapas, a complexidade é depende mais da quantidade de atributos de cada fluxo, a qual pode ser mantida pequena utilizando algum mecanismo para seleção de atributos. Portanto, é possível notar que o EFC apresenta um baixo custo computacional quando comparado a classificadores baseados em ML, como ANN, SVM e RF. Considerando a implementação apresentada nesta seção, a seguir, serão apresentados os resultados obtidos quando o EFC é utilizado para realizar classificação binária de fluxos.

# Capítulo 4

# Metodologia

Neste capítulo, os fluxos de rede serão definidos em maiores detalhes para que possam ser utilizados no EFC. Os *datasets* utilizados neste trabalho serão brevemente descritos na segunda seção. Em seguida, serão apresentados os classificadores baseados ML utilizados nos experimentos comparativos. Os classificadores utilizados como base de comparação para o EFC comparação foram construídos utilizando os seguintes algoritmos de ML: NB, KNN, DT, RF, SVM, Adaboost e ANN. Cada um desses algoritmos será brevemente explicado na terceira seção. Por fim, serão apresentados alguns detalhes dos experimentos realizados.

## 4.1 Definições

Fluxos de rede são conjuntos de pacotes capturados em redes de computadores, caracterizados por diferentes propriedades extraídas após a captura. Um fluxo de rede passa por um determinado ponto de observação na rede durante um intervalo de tempo. Todos os pacotes pertencentes a um mesmo fluxo possuem um conjunto de atributos iguais, chamados chaves de fluxo (*flow keys*), *e.g.,* IP de origem e IP de destino. FlowScan [52] é um exemplo de uma ferramenta capaz de capturar conjuntos de pacotes (fluxos) e extrair seus atributos, que serão posteriormente exportados em diferentes formatos, como NetFlow e IPFIX. A título de exemplificação, uma vez que o NetFlow é o formato mais utilizado, seus principais atributos são listados abaixo:

- IP de origem/destino (*flow key*) - determina a origem e o destino de um determinado fluxo na rede;
- Porta de origem/destino (*flow key*) - caracterizam diferentes tipos de serviços de rede, *e.g.,* a porta 22 é utilizada para acessar um serviço de *ssh*;

- Protocolo (*flow key*) - caracteriza os fluxos em relação ao tipo de protocolo da camada de transporte utilizado, *e.g.,* TCP, UDP, ICMP.
- Número de pacotes (atributo) - número de pacotes capturados em um fluxo;
- Número de bytes (atributo) - número de bytes em um fluxo;
- Duração (atributo) - duração de um fluxo em segundos;
- *Timestamp* inicial (atributo) - tempo de sistema no momento em que o fluxo começa a ser capturado;

Outros atributos como *TCP flags* e tipo de serviço podem também ser exportados em alguns casos. A combinação de diferentes chaves de fluxo e atributos são o que caracteriza um determinado fluxo e o difere de outros.

Abordagens baseadas em fluxos são vistas como boas alternativas para preceder a inspeção de pacotes em NIDSs de tempo real. A ideia é inspecionar profundamente apenas os pacotes pertencentes a fluxos considerados suspeitos pelo classificador de fluxos. Uma abordagem em duas etapas reduziria notavelmente a quantidade de dados a serem analisados, enquanto mantém uma boa acurácia de classificação [15]. Neste trabalho, será tratada apenas a primeira etapa, *i.e.,* classificação de fluxos. O desempenho do algoritmo proposto aqui, o EFC, é comparado ao desempenho de outros algoritmos de ML utilizando três *datasets*. Além disso, foi avaliado o desempenho de todos os algoritmos em casos em que o treino é realizado em uma distribuição de dados e o teste em outra (dentro do mesmo *dataset* e entre *datasets*), caracterizando uma situação de adaptação de domínio (*domain adaptation*). A seguir, serão brevemente descritos os *datasets* utilizados nos testes e será caracterizado o que constitui diferentes domínios em cada um deles.

## 4.2 Datasets

Diversos *datasets* de fluxos de rede compostos por fluxos capturados em diferentes contextos (*e.g.,* em redes de computadores reais ou simuladas) podem ser obtidos em sítios de livre acesso. Portanto, cada *dataset* utilizado neste trabalho é descrito de acordo com a forma como cada um deles foi obtido, bem como a sua composição em termos de tráfego malicioso, *i.e.,* quais tipos de ataque estão presentes e em que proporção. Por fim, define-se a metodologia aplicada para se utilizar cada um dos *datasets* nos experimentos realizados.

### 4.2.1 CIDDS-001

O CIDDS-001 [24] é um *dataset* relativamente recente (2017) composto por um conjunto de amostras de fluxo capturadas em um ambiente de uma rede simulada com OpenStack e outro conjunto de amostras de fluxo capturadas em um servidor real. O primeiro contém apenas tráfego simulado, enquanto o segundo contém tanto tráfego real quanto tráfego simulado. As amostras coletadas em cada um desses dois ambientes pertence a uma das categorias descritas na Tabela 4.1.

Tabela 4.1: Categorias contidas no *dataset* CIDDS-001

| **Ambiente** | **Categorias** |
|---|---|
| OpenStack | normal, DoS, portScan, pingScan, bruteForce |
| External server | normal, DoS, bruteForce, unknown, suspicious |

Fluxos benignos simulados são rotulados como *normal*, enquanto os fluxos maliciosos simulados são rotulados como *dos*, *portScan*, *pingScan* ou *bruteForce*, dependendo do tipo de ataque simulado. As categorias *suspicious* e *unknown*, por sua vez, são utilizadas para tráfego real. O servidor externo é aberto para acesso de usuários por meio das portas 80 e 443. Os fluxos direcionados a essas portas são rotulados como *unknown*, pois eles poderiam ser tanto benignos quanto maliciosos. Por outro lado, todos os fluxos direcionados a outras portas são rotulados como *suspicious*. Neste *dataset*, uma mudança do contexto de tráfego simulado para o contexto de tráfego real foi considerada uma mudança de domínio, utilizada para testar a adaptabilidade dos modelos inferidos pelos classificadores.

Os atributos contidos no *dataset* CIDDS-001 são mostrados na Tabela 4.2. Todos os atributos foram levados em consideração nos testes de classificação, exceto *Src IP*, *Dest IP* e *Date first seen*. Essas exceções são devidas ao fato de que a última intrinsecamente não é informativa para diferenciar fluxos e as primeiras são inventadas no contexto de tráfego simulado, podendo gerar confusão.

### 4.2.2 CICIDS17

O *dataset* CICIDS17 [25] contém tanto tráfego benigno quanto os mais atualizados ataques cibernéticos, similares ao mundo real. Esse *dataset* foi construído simulando o comportamento de 25 usuários baseado nos protocolos HTTP, HTTPS, FTP, SSH, e de email. Os fluxos foram capturados durante uma semana, em julho de 2017. Os ataques implementados incluem *Brute Force FTP*, *Brute Force SSH*, *DoS*, *Heartbleed*, *Web Attack*, *Infiltration*, *Botnet* e *DDoS*. Eles foram executados tanto nas manhãs quanto nas tardes de terça, quarta, quinta e sexta (ver Tabela 4.3).

Tabela 4.2: Atributos contidos no *dataset* CIDDS-001

| # | **Nome** | **Descrição** |
|---|---|---|
| 1 | Src IP | endereço IP de origem |
| 2 | Src Port | porta de origem |
| 3 | Dest IP | endereço IP de destino |
| 4 | Dest Port | porta de destino |
| 5 | Proto | protocolo de transporte (*e.g.,* ICMP, TCP, or UDP) |
| 6 | Date first seen | instante de tempo em que o fluxo se iniciou |
| 7 | Duration | duração do fluxo |
| 8 | Bytes | número de bytes transmitidos |
| 9 | Packets | número de pacotes transmitidos |
| 10 | Flags | concatenação OR de todas *TCP flags* |

Tabela 4.3: Ataques presentes no *dataset* CICIDS17

| **Dia da semana** | **Ataque** |
|---|---|
| Segunda | |
| Terça | FTP-Patator, SSH-Patator |
| Quarta | DoS slowloris, DoS Slowhttptes, DoS Hulk, DoS GoldenEye, Heartbleed Port 444 |
| Quinta | Brute Force, XSS, Sql Injection, Dropbox download, Cool disk |
| Sexta | Botnet ARES, Port Scan, DDoS LOIT |

Os atributos dos fluxos nesse *dataset* foram extraídos utilizando o CICFlowMeter [53]. São mais de 80 atributos no total, que podem ser encontrados no site `http://www.netflowmeter.ca/netflowmeter.html`. Todos os atributos foram considerados neste trabalho, exceto *Flow ID*, *Source IP*, *Destination IP* e *Timestamp*. Essas exceções foram feitas porque esses atributos são intrinsecamente não informativos ou foram inventados no ambiente de simulação.

### 4.2.3 CICDDoS19

O *dataset* CICDDoS19 [26] contém tanto tráfego benigno quanto os ataques DDoS mais atuais (volumétricos e de aplicação: *low volume, slow rate*), similares ao mundo real. Esse *dataset* contém diferentes ataques DDoS de reflexão modernos, como *PortMap*, *NetBIOS*, *LDAP*, *MSSQL*, *UDP*, *UDP-Lag*, *SYN*, *NTP*, *DNS*, e *SNMP*. O tráfego foi

capturado em janeiro (primeiro dia) e março (segundo dia) de 2019. Os ataques foram executados durante esse período (ver Tabela 4.4).

Tabela 4.4: Ataques presentes no *dataset* CICDDoS19

| **Dia** | **Ataques** |
|---|---|
| Primeiro dia | PortMap, NetBIOS, LDAP, MSSQL, UDP, UDP-Lag, SYN |
| Segundo dia | NTP, DNS, LDAP, MSSQL, NetBIOS, SNMP, SSDP, UDP, UDP-Lag, WebDDos, SYN, TFTP |

Os atributos dos fluxos nesse *dataset* foram extraídos utilizando o CICFlowMeter [53]. São mais de 80 atributos no total, que podem ser encontrados no site `http://www.netflowmeter.ca/netflowmeter.html`. Todos os atributos foram considerados neste trabalho, exceto *Flow ID*, *Source IP*, *Destination IP* e *Timestamp*. Essas exceções foram feitas porque esses atributos são intrinsecamente não informativos ou foram inventados no ambiente de simulação. É importante mencionar que os atributos dos *datasets* influenciam de maneira diferente o desempenho dos classificadores de tráfego e precisam ser avaliados. Dessa forma, a seguir, os classificadores utilizados em comparação ao EFC são detalhados.

## 4.3 Classificadores baseados em aprendizagem de máquina

Nesta seção, os classificadores utilizados como base de comparação para o EFC são brevemente descritos. Além disso, são apresentadas a implementação e os parâmetros de cada um deles que foram utilizados na realização dos experimentos de classificação.

### 4.3.1 Naive Bayes

O NB é um classificador estatístico que assume a hipótese de que uma determinada amostra pertence a uma dada classe e, então, calcula a probabilidade de que essa hipótese seja verdadeira. Para tanto, é utilizada uma versão simplificada do modelo de probabilidade Bayesiano [54]. Nesse modelo, é considerada a probabilidade de um resultado dadas várias variáveis. A probabilidade do resultado final é considerada, junto com a probabilidade das variáveis ocorrerem, dado que o resultado final ocorre. Esse modelo assume estatístico assume independência entre as variáveis, o que poderia tornar sua aplicação limitada, porém ele costuma ter bons resultados para a classificação de tráfego de rede [55, 56].

Neste trabalho, foi utilizada a implementação *Gaussian* NB do scikit-learn [57] versão 0.23.2. No *Gaussian* NB, assume-se que a verossimilhança dos atributos seja Gaussiana:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right) \tag{4.1}$$

onde $P(x_i|y)$ é a probabilidade de se observar o atributo $x_i$, dada a classe $y$. O desvio padrão $\sigma_y$ e a média $\mu_y$ são estimados utilizando máxima verossimilhança.

### 4.3.2 K-Nearest Neighbors

O KNN é um algoritmo de mineração de dados comumente utilizado em problemas de classificação de tráfego de rede [58]. A ideia básica por detrás desse algoritmo é a de que, em um espaço de amostras, se os k vizinhos mais próximos a uma amostras pertencem a determinada classe, aquela amostra também vai pertencer [54]. Em um classificador binário de tráfego de rede, a etapa de treino consiste em mapear as amostras de treino em um espaço multidimensional, considerando os diferentes atributos de cada amostra. Por fim, na etapa de teste, as amostras analisadas são projetadas nesse mesmo espaço e verifica-se qual amostra de treino é a mais próxima dela, para determinar a sua classe.

Neste trabalho, foi utilizada a implementação do KNN do scikit-learn [57] versão 0.23.2. Foram utilizados os parâmetros padrões dessa implementação, ou seja, número de vizinhos $k = 5$, função peso uniforme (todos os pontos em cada vizinhança possuem o mesmo peso), $algorithm =' auto'$ (o melhor algoritmo para computar os vizinhos mais próximos de um nodo - *BallTree*, *KDTree* ou *brute force* - é inferido automaticamente com base na entrada), tamanho das folhas = 30 e métrica Minkowski para computar as distâncias, com parâmetro $p = 2$.

### 4.3.3 Decision Tree

O classificador DT [59] é um dos algoritmos de ML mais utilizados no geral, sendo também bastante utilizado para detecção de intrusão em redes. Uma árvore de decisão é composta por nodos de decisão e folhas. Cada nodo de decisão corresponde a um teste sobre um determinado atributo dos dados de entrada e os ramos que saem dele, correspondem aos possíveis resultados do teste. Cada folha representa uma classe, que seria o resultado final de decisão para uma dada amostra [60]. Diferentes critérios podem ser utilizados para decidir qual o melhor atributo para testar em cada nodo durante a construção da árvore, *e.g., Gini impurity* ou *information gain*.

Neste trabalho, utilizamos a implementação do DT do scikit-learn [57] versão 0.23.2 com configuração padrão. O critério utilizado para a construção da árvore é a função

*Gini impurity* e a estratégia para bifurcação de nodos é a melhor (não aleatória). Os demais parâmetros da configuração padrão podem ser consultados na documentação do scikit-learn, disponível no sítio `https://scikit-learn.org`.

### 4.3.4 Random Forest

O RF é um classificador ensemble, composto com um conjunto de árvores de decisão. Esse classificador costuma ter uma menor taxa de erro quando comparado ao DT. Além disso, a combinação de um grande número de árvores de decisão reduz o risco de *overfitting*. Para predizer a classe de uma determinada amostra, o RF considera a predição dada pelo maior número de árvores (esquema de votação). O treino é feito em cada árvore separadamente, podendo ser feito em paralelo. Além disso, existe uma certa aleatoriedade no momento de construção das árvores, que gera diversidade. O RF apresenta um dos melhores desempenhos para classificação de tráfego de rede [61].

Neste trabalho, foi utilizada a implementação do RF do scikit-learn [57] versão 0.23.2 com configuração padrão. Ou seja, o número de árvores utilizadas para a construção da floresta é 100 e é utilizado *bootstrap* de atributos para construção das árvores. Os demais parâmetros da configuração padrão podem ser consultados na documentação do scikit-learn, disponível no sítio `https://scikit-learn.org`.

### 4.3.5 Adaboost

*Boosting* é um termo utilizado para se referir ao problema geral de produzir uma regra de predição acurada a partir da combinação de diferentes regras de predição simples e potencialmente não muito acuradas [62]. O classificador Adaboost é um meta-algoritmo de aprendizagem que começa ajustando um classificador ao *dataset* original. Em seguida, outras cópias desse classificador são ajustadas ao mesmo *dataset*, mas com os pesos de instâncias classificadas incorretamente modificados, de forma que esses novos classificadores estejam focados em resolver os casos mais difíceis ou raros.

Neste trabalho, foi utilizada a implementação do Ababoost do scikit-learn [57] versão 0.23.2 com a configuração padrão. Na configuração padrão, DT é definido como é estimador base, o número de estimados é 50 e a taxa de aprendizagem é 1. Além disso, o algoritmo de *boosting* utilizado é o SAMME.R, uma nova versão do SAMME [63] com convergência mais rápida e baixa taxa de erros.

### 4.3.6 Support Vector Machine

O algoritmo SVM [64] pois possui grande poder preditivo para problemas complexos (multidimensionais). A ideia desse algoritmo é encontrar, por meio de um processo de

otimização, o hiperplano que melhor separa um dado conjunto de treino conforme as classes corretas das amostras. A utilização de uma função kernel torna a separação não-linear do espaço possível. Uma das funções kernel mais comuns é a *radial basis function* (RBF):

$$k(\boldsymbol{x}, \boldsymbol{x'}) = exp(-\gamma||\boldsymbol{x} - \boldsymbol{x'}||^2) \tag{4.2}$$

onde $\gamma > 0$ é um parâmetro, $\boldsymbol{x}$ é uma amostra do conjunto de treino e $\boldsymbol{x'}$ é uma amostra do conjunto de teste.

Neste trabalho, foi utilizada a implementação do SVC (*Support Vector Classifier*) do scikit-learn [57] versão 0.23.2 com a configuração padrão. Os parâmetros padrões desse algoritmo são: parâmetro de regularização $C = 1.0$, kernel radial (*radial basis function kernel*), parâmetro gamma do kernel $\gamma = 1/(n_{features}\sigma_x)$, tolerância = 1e-3 (critério de parada), com heurística de encolhimento (*shrinking heuristic*) e função de decisão *one-vs-rest*. Maiores detalhes podem ser obtidos na documentação do scikit-learn, disponível no sítio: `https://scikit-learn.org`.

### 4.3.7 Artificial Neural Network

ANNs são uma forma de aprendizagem de máquina inspirada pelo funcionamento de neurônios no cérebro humano [65]. Uma ANN é tipicamente composta por duas ou mais camadas de neurônios (ou nodos). As entradas são apresentadas aos neurônios de uma camada inicial, na qual essas informações serão pesadas e processadas, de forma a determinar o resultado que será repassado para as próxima camada, e assim por diante. ANNs utilizam regras de aprendizagem como *gradient descent* e *back-propagation* para propagar erros e vieses através de suas múltiplas camadas, o que as torna capazes de tratar relações complexas e não-lineares entre variáveis [66]. Em problemas de classificação, uma ANN com configuração específica é construída e um conjunto de dados de treinamento é utilizado para atribuir pesos e vieses adequados para cada camada de neurônios. Posteriormente, essa ANN é utilizada para classificar novas amostras.

Neste trabalho, foi utilizado como representante de ANN o classificador *Multi-Layer Perceptron* (MLP) do scikit-learn [57] versão 0.23.2 com configuração padrão. O classificador MLP padrão possui apenas uma camada escondida com 100 neurônios, a função de ativação dos neurônios é a *rectified linear unit function* ($f(x) = max(0, x)$), a função de otimização dos pesos é a 'adam' [67], parâmetro alpha = 0.0001 e taxa de aprendizagem constante = 0.001. Maiores detalhes podem ser obtidos na documentação do scikit-learn, disponível no sítio: `https://scikit-learn.org`.

## 4.4 Desenho dos experimentos de classificação

Nesta seção são apresentados os desenhos dos experimentos realizados neste trabalho. É importante ressaltar que os experimentos de classificação realizados foram desenhados não apenas para avaliar o desempenho de diferentes classificadores, mas também para testar sua capacidade de adaptação a diferentes domínios, *i.e.,* diferentes distribuições de dados. Portanto, foram desenhados dois tipos de experimentos: intra-domínio (treino e teste no mesmo domínio) e inter-domínio (treino em um domínio e teste em outro).

Nos experimentos intra-domínio, para cada um dos três *datasets*, foi computado o desempenho médio de cada classificador sobre dez conjuntos de teste independentes, cada um com 10,000 amostras benignas e 10,000 amostras maliciosas aleatoriamente amostradas do *dataset* completo. Na etapa de treino, foi utilizado 80% do conjunto de teste, enquanto os 20% restantes foram utilizados na etapa de teste. Nos experimentos inter-domínio, os mesmos modelos inferidos na etapa de treino dos testes intra-domínio foram utilizados para avaliar o desempenho médio dos classificadores sobre dez conjuntos contendo 2,000 amostras benignas e 2,000 amostras maliciosas aleatoriamente amostradas de outro domínio (outro contexto do mesmo *dataset* ou outro *dataset*).

### 4.4.1 Classificação binária com o EFC

Para avaliar se o EFC é capaz de classificar corretamente fluxos benignos e maliciosos, foram utilizados dez conjuntos de teste contendo 10,000 amostras benignas e 10,000 amostras maliciosas cada, conforme descrito na introdução desta seção. Todas as amostras classificadas nas dez etapas de teste realizadas foram agrupadas e seus valores de energia foram analisados para verificar se o EFC foi capaz de fazer uma boa separação. Uma análise análoga foi feita considerando os mesmos modelos (já inferidos) e amostras de teste extraídas do tráfego real do *dataset* CIDDS-001. Da mesma forma as energias das amostras consideradas em todas as etapas de teste foram analisadas em conjunto para verificar se houve separação. Outras duas análises análogas foram feitas considerando os *datasets* CICIDS17 e CICDDoS19. No entanto, o teste inter-domínio do CICIDS17 foi feito considerando amostras extraídas do CICDDoS19 e vice-versa. Ou seja, como os atributos presentes nesses dois *datasets* coincidem, os experimentos inter-domínio foram feitos entre os *datasets*.

A Tabela 4.5 mostra as classes consideradas para a discretização dos atributos presentes no *dataset* CIDDS-001. Uma vez que *TCP Flags* é o atributo discreto com a maior qunatidade de valores possíveis (32 possibilidades), o tamanho do alfabeto $Q$ foi definido como 32 para esse *dataset*. Cada atributo contínuo foi, portanto, discretizado em um certo número de classes (ou *bins*) maior ou igual a 32. As classes foram determinadas de

forma que a quantidade de valores pertencentes a cada classes fosse similar entre todas as classes. Os atributos presentes nos *datasets* CICIDS17 e CICDDoS19 também foram discretizados de forma que a quantidade de valores em cada classe fosse similar para todas as classes. Essas discretizações não são mostradas aqui por conta do grande número de atributos presentes nesses *datasets* (>80 atributos).

Tabela 4.5: Classes consideradas para discretização dos atributos no *dataset* CIDDS-001

| Feature | List of classes upper limits |
|---|---|
| Duration | 0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.01, 0.04, 1, 10, 100, $\infty$ |
| Protocol | TCP, UDP, GRE, ICMP, IGMP |
| Src Port | 50, 60, 100, 400, 500, 40000, 60000, $\infty$ |
| Dst Port | 50, 60, 100, 400, 500, 40000, 60000, $\infty$ |
| Num. of Bytes | 50, 60, 70, 90, 100, 110, 200, 300, 400, 500, 700, 1000, 5000, $\infty$ |
| Num. of Packets | 2, 3, 4, 5, 6, 7, 10, 20, $\infty$ |
| TCP Flags | $\{(f_1, f_2, f_3, f_4, f_5) \mid f_i \in \{0, 1\}\}$ |

### 4.4.2 Análise comparativa do desempenho do EFC

O EFC foi comparado a diferentes classificadores de fluxos baseado em ML: K-Nearest Neighbors (KNN) [68], DT [59, 69], Adaboost [70], Random Forest (RF) [71], ANN [72], NB [73], e SVM [64], todos com as configurações padrão da biblioteca *scikit-learn*. Cabe ressaltar que os métodos *ensemble*, *i.e.,* RF e Adaboost foram considerados separadamente dos outros métodos, que são classificadores simples. Essa decisão foi tomada após contatar que o DT, classificador base do RF e Adaboost, obtém um F1-score acima de 0.99 em todos os experimentos intra-domínio. Os atributos foram discretizados apenas no caso do EFC (Tabela 4.5), uma vez que a discretização poderia prejudicar o desempenho dos outros classificadores.

As métricas utilizadas para comparar os resultados foram o F1-score e a área sobre a curva ROC (AUC). A primeira métrica, F1-score, é a média harmônica das métricas *Precision* e *Recall*, *i.e.,*

$$F1 = \frac{2}{Precision^{-1} + Recall^{-1}} = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{4.3}$$

onde $Precision = TP/(TP + FP)$, $Recall = TP/(TP + FN)$, TP é o número de verdadeiros positivos, *i.e.,* fluxos maliciosos classificados como maliciosos, FP é o número de falsos positivos, *i.e.,* fluxos benignos classificados como maliciosos, e FN é o número de falsos negativos, *i.e.,* o número de fluxos maliciosos classificados como benignos.

A segunda métrica, área sobre a curva ROC (AUC), é uma das métricas mais utilizadas para classificadores binários [74, 75]. A curva ROC é construída plotando a taxa de verdadeiros positivos (TPR) contra a taxa de falsos positivos (FPR) para diferentes limiares de classificação. Isso significa que a AUC representa a probabilidade de que um exemplo positivo aleatoriamente escolhido receba uma pontuação maior que um exemplo negativo aleatoriamente escolhido. Uma das maiores vantagens da AUC é que ela é invariante em relação a mudanças nas distribuições das classes, *i.e.,* a curva ROC não muda se a distribuição muda em um conjunto de teste, mas as distribuições condicionais subjacentes de onde os dados foram amostrados é a mesma [76, 75]. Como estamos interessados em investigar a capacidade de adaptação a diferentes domínios dos classificadores, essa métrica é especialmente interessante para este trabalho.

Para avaliar o desempenho do EFC comparado a outros classificadores, foram realizados três experimentos independentes. O primeiro experimento foi realizado sobre o *dataset* CIDDS-001. A etapa de treino foi realizada com amostras de fluxos de rede de tráfego simulado, enquanto a etapa de teste foi realizada considerando tanto amostras de fluxos de tráfego simulado quando de tráfego real, capturadas em um servidor externo. O segundo e o terceiro experimentos foram experimentos inter-*dataset* realizados sobre o CICIDS17 e o CICDDoS19. No segundo experimento, o treino foi realizado com o CICIDS17, enquanto o teste foi realizado sobre os dois *datasets*. No terceiro experimento, o treino foi realizado com o CICDDoS19, enquando o teste foi realizado sobre os dois *datasets*.

Essencialmente, em cada experimento, foi mensurado o desempenho dos classificadores quando treinados e testados no mesmo domínio e quando treinado em um domínio e testado em um domínio diferente. O desempenho foi mensurado com base na média dos valores de AUC e F1-score sobre dez conjuntos de teste e no erro padrão, com intervalo de confiança de 95%. O limiar de classificação do EFC foi definido como o 95% percentil da distribuição de energia obtida na etapa de treino, ou seja, a definição do limiar, assim como o treino, é baseada apenas em amostras benignas. A composição média dos conjuntos de teste são mostradas nas Tabelas 4.6 e 4.7.

Tabela 4.6: Composição média dos conjuntos de teste utilizados no experimento 1

| CIDDS-001 OpenStack | | CIDDS-001 real traffic | |
|---|---|---|---|
| Classe | Número | Classe | Número |
| *normal* | 10,000 | *unknown* | 2,000 |
| *dos* | 9,800 | *suspicious* | 2,000 |
| *pingScan* | 20 | | |
| *portScan* | 150 | | |
| *bruteForce* | 30 | | |
| **Total** | 20,000 | **Total** | 4,000 |

Tabela 4.7: Composição média dos conjuntos de teste utilizados nos experimentos 2 e 3

| CICIDS17 | | CICDDoS19 | |
|---|---|---|---|
| Classe | Número | Classe | Número |
| *benign* | 10,000 | *benign* | 10,000 |
| *FTP Patator* | 170 | *DrDoS DNS* | 890 |
| *SSH Patator* | 80 | *DrDoS LDAP* | 370 |
| *DDoS* | 2,740 | *DrDoS MSSQL* | 890 |
| *PortScan* | 1,060 | *DrDoS NetBIOS* | 880 |
| *Bot* | 110 | *DrDoS NTP* | 890 |
| *Infiltration* | 20 | *DrDoS SNMP* | 200 |
| *Brute force* | 50 | *DrDoS SSDP* | 890 |
| *SQL injection* | 10 | *DrDoS UDP* | 890 |
| *XSS* | 10 | *Syn* | 890 |
| *DoS Hulk* | 2,730 | *TFTP* | 890 |
| *DoS GoldenEye* | 2,730 | *LDAP* | 120 |
| *DoS Slowloris* | 120 | *NetBIOS* | 140 |
| *DoS Slowhttptest* | 170 | *MSSQL* | 660 |
| | | *Portmap* | 400 |
| | | *UDP* | 880 |
| | | *UDPLag* | 120 |
| **Total** | 20,000 | **Total** | 20,000 |

# Capítulo 5

# Resultados

Nesta seção, são apresentados os resultados obtidos para o EFC e classificadores baseados em ML em diferentes experimentos de classificação binária considerando três *datasets* distintos: CIDDS-001, CICIDS17 e CICDDoS19. Primeiramente, será mostrado que o EFC é capaz de separar fluxos benignos de fluxos maliciosos com base em valores de energias, um resultado que se mostra consistente em todos os *datasets*. Em seguida, são apresentados os resultados de diferentes experimentos de classificação binária de fluxos e o desempenho do EFC é comparado ao desempenho de classificadores baseados em ML, considerando treino/teste no mesmo domínio (experimentos intra-domínio) e treino/teste em domínios distintos (experimentos inter-domínio).

## 5.1 Classificação binária com o EFC

Para investigar se o EFC é capaz de classificar corretamente fluxos de rede benignos e maliciosos, foi inferido um modelo estatístico baseado em amostras de fluxos benignos do tráfego simulado do *dataset* CIDDS-001. Esse modelo foi então utilizado para calcular as energias de amostras de fluxos benignos e maliciosos vindas tanto do tráfego simulado quando do tráfego real, capturado em um servidor externo, conforme descrito na seção 4.4.1 do capítulo anterior. A Figura 5.1A mostra os valores de energia de 40,000 amostras de fluxos, a união dos resultados obtidos considerando todos dos dez conjuntos de teste. O modelo estatístico utilizado para calcular as energias em cada conjunto de teste foi inferido com base em 8,000 amostras de fluxos benignas selecionadas aleatoriamente do conjunto completo de amostras tráfego simulado. A separação entre as duas classes de fluxos é clara, *i.e.,* a energia de fluxos benignos está claramente deslocada para a esquerda em relação à distribuição das energias de fluxos maliciosos.

As energias de 20,000 fluxos classificados como *unknown* e 20,000 fluxos classificados como *suspicious* aleatoriamente amostrados do tráfego real capturado no servidor externo

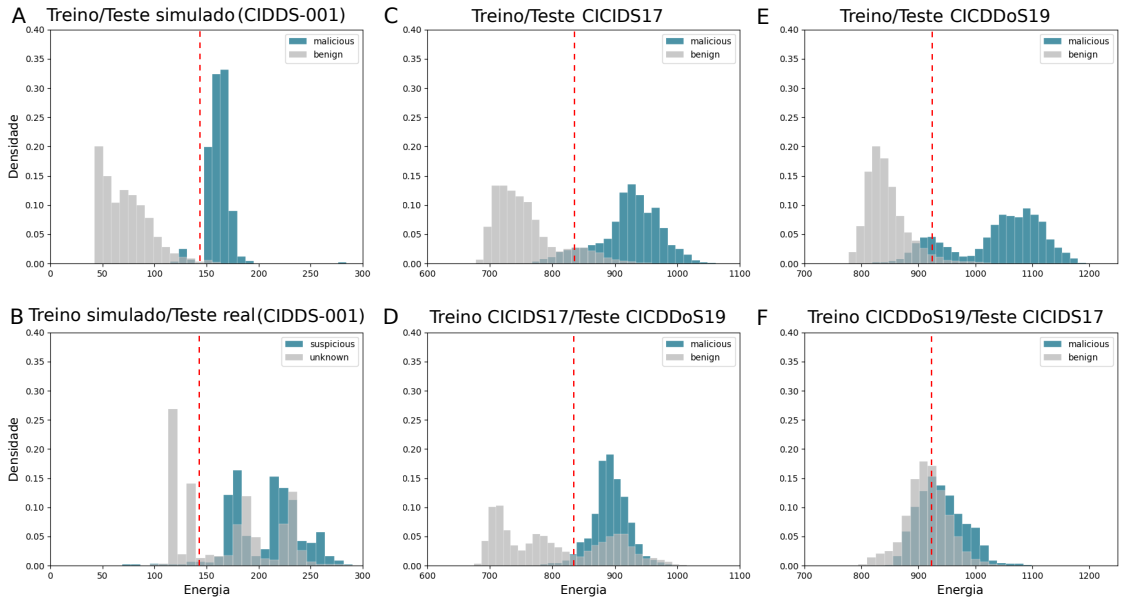


Figura 5.1: Histogramas das energias de fluxos benignos (n = 20,000 em cada *plot*) e maliciosos (n = 20,000 em cada *plot*) calculadas na fase de teste de um experimento de classificação realizado sobre o *dataset* CIDDS-001 (A,B), e dois experimentos inter-*dataset* realizados sobre os *datasets* CICIDS17 e CICDDoS19 (C-F).

do *dataset* CIDDS-001 são mostradas na Figura 5.1B. De acordo com as definições do *dataset*, o tráfego rotulado como *unknown* é o tráfego que chega nas portas esperadas do servidor (porta 80 e porta 443). Portanto, esse tráfego pode ser tanto benigno quanto malicioso. Já o tráfego rotulado com *suspicious*, é o tráfego inesperado que chega em portas do servidor que não estão abertas. Portanto, esse tráfego é considerado malicioso. É importante ressaltar que a separação entre essas duas classes não é tão evidente, uma vez que parte do tráfego rotulado como *unknown* se mistura o tráfego rotulado como *suspicious*. Entretanto, é possível notar que existe uma porção distinta do tráfego *unknown* que tende a ter uma energia menor que o restante e pode ser claramente distinguida. Visivelmente, é possível aplicar o mesmo limiar de energia (linha vermelha tracejada) para separar tráfego benigno e malicioso nos dois ambientes: simulado e real. Como o limiar aprendido no ambiente simulado pode ser aplicado para diferenciar o tráfego do ambiente real, pode-se concluir que o modelo inferido funcionou corretamente em diferentes domínios, o que corrobora a hipótese de que o modelo seja resiliente a pequenas mudanças na distribuição de dados.

A Figura 5.1C-F mostra os resultados de uma análise inter-*dataset* realizada sobre os *datasets* CICIDS17 e CICDDoS19. Quando treinado com o CICIDS17, o EFC é capaz de separar claramente as classes tanto nesse *dataset* (Figura 5.1C), quanto no CICDDoS19 (Figura 5.1D). Além disso, nota-se que é possível encontrar um limiar único para diferenciar entre tráfego malicioso e benigno, o que reforça a conclusão de que o modelo inferido

pelo EFC é aplicável a diferentes domínios. Entretanto, quando treinado com o *dataset* CICDDoS19, o EFC é capaz de separar as classes nesse *dataset* (Figura 5.1E), porém não apresenta um desempenho tão bom no CICIDS17 (Figura 5.1F). Isso acontece possivelmente devido ao fato de que os tipos de ataques presentes no CICDDoS19 são muito menos diversos, restritos a DDoS, do que os tipos de ataque presentes no CICIDS17, que cobre um maior espectro de classes de ataques. Entretanto, mesmo que as distribuições se sobreponham bastante, ainda assim é possível encontrar um limiar que funcione para os dois casos, conforme indicado pela linha vermelha tracejada.

Em resumo, os resultados apresentados nesta subseção mostram que o EFC é capaz de discriminar corretamente amostras de fluxo em duas categorias, *i.e.,* benignos e maliciosos, e os resultados obtidos se mantém para diferentes *datasets*. Além disso, foi observado que o limiar de energia para classificação, definido com base na distribuição de energias de treino, pode ser aplicado a diferentes distribuições de dados, *i.e.,* domínios. Isso é uma evidência de que, na maioria dos casos, o EFC gera um modelo resiliente, que pode ser aplicado a diferentes domínios e não apresenta problemas de *overfit*. Na próxima subseção, serão apresentados os resultados da classificação binária de fluxos utilizando diferentes classificadores baseados em ML, além do EFC.

## 5.2 Análise comparativa do desempenho do EFC

Para avaliar o desempenho do EFC comparado a diferentes classificadores baseados em ML foram realizados três experimentos independentes. No primeiro experimento, foram considerados dois cenários distintos presentes no *dataset* CIDDS-001, um cenário de tráfego simulado e um cenário de tráfego real. O segundo e o terceiro experimentos, por sua vez, foram experimentos realizados entre os *datasets* CICIDS17 e CICDDoS19, com treino realizado no primeiro e teste no segundo, e vice-versa. O desempenho dos classificadores foi medido em termos de F1-*score* e AUC médios, mais ou menos o erro padrão (IC = 95%). Os resultados são apresentados a seguir.

A Tabela 5.1 mostra o desempenho médio e erro padrão (com intervalo de confiança de 95%) de cada classificador simples no primeiro experimento, realizado sobre o *dataset* CIDDS-001. Na situação em que treino e teste foram realizados com tráfego simulado, DT foi o algoritmo que apresentou o melhor desempenho, com um F1-*score* de $0.999 \pm 0.000$ e AUC de $0.999 \pm 0.000$. O EFC também apresentou um bom desempenho, sendo o segundo melhor em termos de AUC ($0.997 \pm 0.001$). Já na situação em que o treino foi realizado com tráfego simulado e o teste com tráfego real, o EFC superou os demais classificadores em F1-*score* ($0.675 \pm 0.009$) e AUC ($0.720 \pm 0.001$). É notável que todos os classificadores tiveram uma considerável perda de desempenho quando testados em

um domínio diferente, mostrando a sensibilidade dos modelos inferidos a mudanças na distribuição de dados.

Tabela 5.1: Experimento 1 (simples) - desempenho médio e erro padrão (IC = 95%) com treino realizado sobre o tráfego simulado do *dataset* CIDDS-001

| | Treino/Teste simulado | | Treino simulado/Teste real | |
|---|---|---|---|---|
| Classificador | F1 score | AUC | F1 score | AUC |
| NB | 0.043 ± 0.016 | 0.502 ± 0.004 | 0.057 ± 0.024 | 0.517 ± 0.016 |
| KNN | 0.988 ± 0.001 | 0.994 ± 0.000 | 0.118 ± 0.014 | 0.524 ± 0.001 |
| DT | **0.999 ± 0.000** | **0.999 ± 0.000** | 0.556 ± 0.007 | 0.619 ± 0.000 |
| SVM | 0.805 ± 0.003 | 0.951 ± 0.002 | 0.531 ± 0.005 | 0.707 ± 0.003 |
| MLP | 0.979 ± 0.002 | 0.993 ± 0.001 | 0.151 ± 0.016 | 0.596 ± 0.002 |
| **EFC** | 0.975 ± 0.001 | 0.997 ± 0.001 | **0.675 ± 0.009** | **0.720 ± 0.001** |

Analogamente, a Tabela 5.2 mostra os resultados obtidos pelos classificadores *ensemble* no primeiro experimento. Tanto o RF quanto o Adaboost alcançam um desempenho quase perfeito no teste intra-domínio (F1-*score* = $0.999 \pm 0.000$ e AUC = $1.000 \pm 0.000$), mostrando que esse *dataset* não representa um desafio para esses classificadores. No teste inter-domínio, entretanto, há uma queda considerável no desempenho em ambos os casos. Curiosamente, nenhum deles conseguiu alcançar o mesmo desempenho do EFC (F1-*score* = $0.675 \pm 0.009$ e AUC = $0.720 \pm 0.001$).

Tabela 5.2: Experimento 1 - (*ensemble*) - desempenho médio e erro padrão (IC = 95%) com treino realizado sobre o tráfego simulado do *dataset* CIDDS-001

| | Treino/Teste simulado | | Treino simulado/Teste real | |
|---|---|---|---|---|
| Classificador | F1 score | AUC | F1 score | AUC |
| Adaboost | 0.999 ± 0.000 | 1.000 ± 0.000 | 0.594 ± 0.022 | 0.630 ± 0.000 |
| RF | 0.999 ± 0.000 | 1.000 ± 0.000 | 0.269 ± 0.018 | 0.714 ± 0.000 |

A Tabela 5.3 mostra os resultados do segundo experimento, realizado sobre os *datasets* CICIDS17 e CICDDoS19, considerando classificadores simples. Na situação em que o treino e o teste foram realizados sobre o CICIDS17, DT foi novamente o algoritmo que apresentou melhor desempenho em termos de F1-*score* ($0.994 \pm 0.001$) e de AUC ($0.994 \pm 0.001$), apesar de seu valor de AUC ser indistinguível do valor de AUC obtido pelo MLP ($0.993 \pm 0.001$). Notavelmente, quando treinados com o CICIDS17 e testados com o CICDDoS19, o EFC mais uma vez superou os outros classificadores em termos de F1-*score* ($0.787 \pm 0.004$) e de AUC ($0.781 \pm 0.003$). Novamente, os resultados mostram que o

EFC é o algoritmo menos sensível a mudanças de domínio considerando as duas métricas avaliadas.

Tabela 5.3: Experimento 2 - (simples) - desempenho médio e erro padrão (IC = 95%) com treino realizado sobre o *dataset* CICIDS17

| | Treino/Teste CICIDS17 | | Treino CICIDS17/Teste CICDDoS19 | |
|---|---|---|---|---|
| Classificador | F1-score | AUC | F1-score | AUC |
| NB | 0.344 ± 0.049 | 0.413 ± 0.021 | 0.344 ± 0.049 | 0.413 ± 0.049 |
| KNN | 0.961 ± 0.001 | 0.987 ± 0.001 | 0.457 ± 0.046 | 0.771 ± 0.001 |
| DT | **0.994 ± 0.001** | **0.994 ± 0.001** | 0.168 ± 0.090 | 0.525 ± 0.001 |
| SVM | 0.930 ± 0.003 | 0.974 ± 0.001 | 0.264 ± 0.025 | 0.664 ± 0.003 |
| MLP | 0.961 ± 0.003 | **0.993 ± 0.001** | 0.221 ± 0.033 | 0.775 ± 0.003 |
| **EFC** | 0.898 ± 0.003 | 0.975 ± 0.001 | **0.787 ± 0.004** | **0.781 ± 0.003** |

Analogamente, a Tabela 5.4 mostra os resultados do segundo experimento, considerando os classificadores *ensemble*. Apesar de os classificadores terem tido uma maior dificuldade de ajustar o limiar para este *dataset*, obtendo valores de F1-*score* de $0.991 \pm 0.002$ e $0.997 \pm 0.001$, o CICIDS17 também parece não ter sido desafiador, visto que ambos os classificadores obtiveram um valor de AUC perfeito ($1.000 \pm 0.000$) no experimento intra-domínio. Já no experimento inter-domínio, mais uma vez houve queda significativa no desempenho. Desta vez, entretanto, o RF obteve um valor de AUC ($0.867 \pm 0.001$) superior ao do EFC (AUC $= 0.781 \pm 0.003$), apesar do F1-*score* ($0.021 \pm 0.003$) muito baixo.

Tabela 5.4: Experimento 2 (*ensemble*) - desempenho médio e erro padrão (IC = 95%) com treino realizado sobre o *dataset* CICIDS17

| | Treino/Teste CICIDS17 | | Treino CICIDS17/Teste CICDDoS19 | |
|---|---|---|---|---|
| Classificador | F1-score | AUC | F1-score | AUC |
| Adaboost | 0.991 ± 0.002 | 1.000 ± 0.000 | 0.228 ± 0.055 | 0.698 ± 0.002 |
| RF | 0.997 ± 0.001 | 1.000 ± 0.000 | 0.021 ± 0.003 | 0.867 ± 0.001 |

Por fim, a Tabela 5.5 mostra os resultados do terceiro experimento, realizado sobre os *datasets* CICIDS17 e CICDDoS19, considerando classificadores simples. Mais uma vez, DT superou os outros classificadores quando treino e teste foram realizados sobre o mesmo *dataset*, com F1-*score* de $0.998 \pm 0.000$ e AUC de $0.998 \pm 0.000$. Quando testados no *dataset* CICIDS17, entretanto, o EFC obteve o melhor valor de F1-*score* ($0.641 \pm 0.002$), enquanto o KNN foi o melhor em termos de AUC ($0.670 \pm 0.002$). A AUC do EFC ($0.664 \pm 0.002$) foi a segunda melhor, o que significa que o desempenho do EFC foi o melhor, levando em

consideração as duas métricas em conjunto. Apesar da adaptação ser mais complicada nesse caso, o desempenho do EFC foi consistente em todos os experimentos realizados.

Tabela 5.5: Experimento 3 (simples) - desempenho médio e erro padrão (IC = 95%) com treino realizado sobre o *dataset* CICDDoS19

| | Treino/Teste CICDDoS19 | | Treino CICDDoS19/Teste CICIDS17 | |
|---|---|---|---|---|
| Classificador | F1-score | AUC | F1-score | AUC |
| NB | 0.590 ± 0.006 | 0.428 ± 0.007 | 0.590 ± 0.006 | 0.428 ± 0.006 |
| KNN | 0.960 ± 0.002 | 0.984 ± 0.001 | 0.397 ± 0.043 | **0.670 ± 0.002** |
| DT | **0.998 ± 0.000** | **0.998 ± 0.000** | 0.259 ± 0.012 | 0.476 ± 0.000 |
| SVM | 0.933 ± 0.002 | 0.976 ± 0.002 | 0.239 ± 0.009 | 0.538 ± 0.002 |
| MLP | 0.968 ± 0.002 | 0.993 ± 0.001 | 0.227 ± 0.011 | 0.451 ± 0.002 |
| **EFC** | 0.916 ± 0.002 | 0.981 ± 0.001 | **0.641 ± 0.002** | 0.664 ± 0.002 |

Analogamente, a Tabela 5.6 mostra os resultados do terceiro experimento, considerando classificadores *ensemble*. Mais uma vez, no experimento intra-domínio, os classificadores não tiveram dificuldade em classificar perfeitamente o conjunto de teste, ambos com AUC = $1.000 \pm 0.000$. No experimento inter-domínio, desta vez foi o Adaboost que obteve melhor desempenho (F1-*score* = $0.270 \pm 0.013$ e AUC = $0.660 \pm 0.001$), mas não chegou a superar o EFC (F1*score* = $0.641 \pm 0.002$ e AUC = $0.664 \pm 0.002$).

Tabela 5.6: Experimento 3 (*ensemble*) - desempenho médio e erro padrão (IC = 95%) com treino realizado sobre o *dataset* CICDDoS19

| | Treino/Teste CICDDoS19 | | Treino CICDDoS19/Teste CICIDS17 | |
|---|---|---|---|---|
| Classificador | F1-score | AUC | F1-score | AUC |
| Adaboost | 0.995 ± 0.001 | 1.000 ± 0.000 | 0.270 ± 0.013 | 0.660 ± 0.001 |
| RF | 0.997 ± 0.000 | 1.000 ± 0.000 | 0.089 ± 0.032 | 0.623 ± 0.000 |

No geral, os resultados apresentados nesta subseção mostram que o EFC é capaz de se adaptar melhor a diferentes domínios do que os classificadores baseados em algoritmos clássicos de ML. Além disso, quando o treino e o teste são realizados no mesmo domínio, o EFC atinge valores de AUC similares aos valores atingidos pelos melhores algoritmos de ML, mostrando que ele é capaz de separar bem as duas classes de fluxos, mesmo sendo treinado com metade da informação apresentada aos outros classificadores. De fato, não utilizar amostras maliciosas na etapa de treino é provavelmente a razão pela qual o EFC é tão bom em se adaptar a outros domínios. A maior resiliência do EFC a mudanças significativas nas distribuições de dados é uma característica muito desejável em classificadores de fluxos de rede, uma vez que mudanças na composição do tráfego são

esperadas e muito frequentes, com novos tipos de ataques sendo continuamente gerados. No próximo capítulo, serão apresentadas as conclusões e possíveis desdobramentos deste trabalho.

# Capítulo 6

# Conclusões

Neste trabalho, foi apresentado um novo classificador baseado em fluxos para detecção de intrusão em rede, chamado *Energy-based Flow Classifier* (EFC). Na etapa de treino do EFC, um modelo estatístico é inferido com base em amostras de tráfego benigno apenas. Em seguida, esse modelo estatístico é utilizado para para classificar fluxos de rede em benignos ou maliciosos com base em valores de energia. Os resultados obtidos mostram que o EFC é capaz de fazer corretamente a classificação binária de fluxos de rede, considerando três *datasets* diferentes. Os valores de F1-score (97% no melhor caso) e AUC (99% no melhor caso) obtidos com o EFC são comparáveis aos valores dessas métricas obtidos com outros classificadores baseados em ML, como *Decision Tree* e *Multi-Layer Perceptron*, mesmo utilizando apenas metade da informação na fase de treino.

Além disso, foi analisada a capacidade de adaptação a novos domínios dos mesmos classificadores e foi observado que o EFC é menos sensível a pequenas mudanças nas distribuições de dados do os outros algoritmos. Em todos os experimentos inter-domínios realizados, o EFC superou os demais classificadores. É provável que a capacidade do EFC de se adaptar a diferentes domínios esteja relacionada ao fato de que a inferência do modelo estatístico é feita com base apenas em amostras benignas, o que previne a ocorrência de *overfitting*.

Considerando os resultados apresentados no capítulo anterior, o EFC parece ser um algoritmo promissor para realização de classificação de tráfego de rede baseada em fluxos. Entretanto, apesar dos resultados serem promissores, há espaço para futuros testes e melhorias. Em trabalhos futuros, o objetivo é realizar investigações mais abrangentes sobre a aplicabilidade do EFC a tráfego do mundo real e também em outros contextos que vão além da detecção de intrusão em rede, como, por exemplo, para detecção de fraudes em dados bancários.

# Referências


[1] Esposito, M, C Mazzariello, F Oliviero, L Peluso, SP Romano e C Sansone: *Intrusion detection and reaction: an integrated approach to network security*. Intrusion Detection Systems, ed, páginas 172–210, 2008. ix, 3

[2] Symantec: *Internet Security Threat Report (ISTR) 2019 | Symantec*, abril 2019. `https://www.symantec.com/security-center/threat-report`, acesso em 2019-05-16. 1

[3] Pradhan, Manoranjan, Chinmaya Kumar Nayak e Sateesh Kumar Pradhan: *Intrusion detection system (ids) and their types*. Em *Securing the Internet of Things: Concepts, Methodologies, Tools, and Applications*, páginas 481–497. IGI Global, 2020. 1

[4] Bruneau, Guy: *The history and evolution of intrusion detection*. SANS Institute, 1, 2001. 1

[5] Anderson, James P: *Computer security threat monitoring and surveillance*. Technical Report, James P. Anderson Company, 1980. 1

[6] Denning, Dorothy E, DL Edwards, R Jagannathan, TF Lunt e Peter G Neumann: *A prototype ides: A real-time intrusiondetection expert system*. Computer Science Laboratory, SRI International, 1987. 1

[7] Mukherjee, Biswanath, L Todd Heberlein e Karl N Levitt: *Network intrusion detection*. IEEE network, 8(3):26–41, 1994. 2

[8] Di Pietro, Roberto e Luigi V Mancini: *Intrusion detection systems*, volume 38. Springer Science & Business Media, 2008. 2

[9] Liao, Hung Jen, Chun Hung Richard Lin, Ying Chih Lin e Kuang Yuan Tung: *Intrusion detection system: A comprehensive review*. Journal of Network and Computer Applications, 36(1):16–24, 2013. 2, 3

[10] Debar, Hervé, Marc Dacier e Andreas Wespi: *A revised taxonomy for intrusion-detection systems*. Em *Annales des télécommunications*, volume 55, páginas 361–378. Springer, 2000. 3

[11] Bolzoni, Damiano e Sandro Etalle: *Approaches in anomaly-based network intrusion detection systems*. Intrusion Detection Systems, 38:1–15, 2008. 3, 4, 5

[12] Tavallaee, Mahbod, Ebrahim Bagheri, Wei Lu e Ali A Ghorbani: *A detailed analysis of the kdd cup 99 data set*. Em *2009 IEEE symposium on computational intelligence for security and defense applications*, páginas 1–6. IEEE, 2009. 4

[13] Galassi, Ugo: *Learning behavior profiles from noisy sequences*. Intrusion Detection Systems, 38:39–64, 2008. 5

[14] Ring, Markus, Sarah Wunderlich, Deniz Scheuring, Dieter Landes e Andreas Hotho: *A survey of network-based intrusion detection data sets*. Computers & Security, 2019. 5

[15] Sperotto, Anna, Gregor Schaffrath, Ramin Sadre, Cristian Morariu, Aiko Pras e Burkhard Stiller: *An overview of IP flow-based intrusion detection*. IEEE Communications Surveys and Tutorials, 12(3):343–356, 2010, ISSN 1553877X. 5, 20

[16] Umer, Muhammad Fahad, Muhammad Sher e Yaxin Bi: *Flow-based intrusion detection: Techniques and challenges*. Computers and Security, 70:238–254, sep 2017, ISSN 01674048. 5, 7

[17] Singla, Ankush, Elisa Bertino e Dinesh Verma: *Overcoming the lack of labeled data: training intrusion detection models using transfer learning*. Em *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*, páginas 69–74. IEEE, 2019. 5, 10

[18] Lu, Jie, Anjin Liu, Fan Dong, Feng Gu, Joao Gama e Guangquan Zhang: *Learning under concept drift: A review*. IEEE Transactions on Knowledge and Data Engineering, 31(12):2346–2363, 2018. 5

[19] Zolanvari, Maede, Marcio A Teixeira e Raj Jain: *Effect of imbalanced datasets on security of industrial iot using machine learning*. Em *2018 IEEE International Conference on Intelligence and Security Informatics (ISI)*, páginas 112–117. IEEE, 2018. 5, 10

[20] Bartos, Karel, Michal Sofka e Vojtech Franc: *Optimized invariant representation of network traffic for detecting unseen malware variants*. Em *25th {USENIX} Security Symposium ({USENIX} Security 16)*, páginas 807–822, 2016. 5, 10

[21] Li, Hao, Zhenxiang Chen, Riccardo Spolaor, Qiben Yan, Chuan Zhao e Bo Yang: *Dart: Detecting unseen malware variants using adaptation regularization transfer learning*. Em *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, páginas 1–6. IEEE, 2019. 5, 10

[22] Rudin, Cynthia: *Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead*. Nature Machine Intelligence, 1(5):206–215, 2019. 5

[23] Holzinger, Andreas: *From machine learning to explainable ai*. Em *2018 World Symposium on Digital Intelligence for Systems and Machines (DISA)*, páginas 55–66. IEEE, 2018. 5, 10

[24] 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. 6, 21

[25] Sharafaldin, Iman, Arash Habibi Lashkari e Ali A Ghorbani: *Toward generating a new intrusion detection dataset and intrusion traffic characterization.* Em *ICISSP*, páginas 108–116, 2018. 6, 21

[26] Sharafaldin, Iman, Arash Habibi Lashkari, Saqib Hakak e Ali A Ghorbani: *Developing realistic distributed denial of service (ddos) attack dataset and taxonomy.* Em *2019 International Carnahan Conference on Security Technology (ICCST)*, páginas 1–8. IEEE, 2019. 6, 22

[27] Vinayakumar, R, KP Soman e Prabaharan Poornachandran: *Evaluating effectiveness of shallow and deep networks to intrusion detection system.* Em *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, páginas 1282–1289. IEEE, 2017. 7, 8

[28] Mahfouz, Ahmed M, Deepak Venugopal e Sajjan G Shiva: *Comparative analysis of ml classifiers for network intrusion detection.* Em *Fourth International Congress on Information and Communication Technology*, páginas 193–207. Springer, 2020. 7, 8

[29] Khan, Sharfuddin, E Sivaraman e Prasad B Honnavalli: *Performance evaluation of advanced machine learning algorithms for network intrusion detection system.* Em *Proceedings of International Conference on IoT Inclusive Life (ICIIL 2019), NITTTR Chandigarh, India*, páginas 51–59. Springer, 2020. 7, 8

[30] Moustafa, Nour, Benjamin Turnbull e Kim Kwang Raymond Choo: *An ensemble intrusion detection technique based on proposed statistical flow features for protecting network traffic of internet of things.* IEEE Internet of Things Journal, 2018. 7, 8

[31] Tama, Bayu Adhi e Kyung Hyune Rhee: *Attack classification analysis of iot network via deep learning approach.* Research Briefs on Information & Communication Technology Evolution (ReBICTE), 3:1–9, 2017. 7, 8

[32] Idhammad, Mohamed, Karim Afdel e Mustapha Belouch: *Detection system of http ddos attacks in a cloud environment based on information theoretic entropy and random forest.* Security and Communication Networks, 2018, 2018. 7, 8

[33] Idhammad, Mohamed, Karim Afdel e Mustapha Belouch: *Distributed intrusion detection system for cloud environments based on data mining techniques.* Procedia Computer Science, 127:35–41, 2018. 7, 8

[34] Tan, Xiaopeng, Shaojing Su, Zhiping Huang, Xiaojun Guo, Zhen Zuo, Xiaoyong Sun e Longqing Li: *Wireless sensor networks intrusion detection based on smote and the random forest algorithm.* Sensors, 19(1):203, 2019. 8

[35] Kazemitabar, J, R TAHERI e GH KHERADMANDIAN: *A novel technique for improvement of intrusion detection via combining random forrest and genetic algorithm.* 2019. 8

[36] Bhavani, T Tulasi, M Kameswara Rao e A Manohar Reddy: *Network intrusion detection system using random forest and decision tree machine learning techniques.* Em *First International Conference on Sustainable Technologies for Computational Intelligence*, páginas 637–643. Springer, 2020. 8

[37] Verma, Abhishek e Virender Ranga: *Statistical analysis of cidds-001 dataset for network intrusion detection systems using distance-based machine learning.* Procedia Computer Science, 125:709–716, 2018. 9

[38] Ring, Markus, Dieter Landes e Andreas Hotho: *Detection of slow port scans in flow-based network traffic.* PloS one, 13(9):e0204507, 2018. 9

[39] Abdulhammed, Razan, Miad Faezipour, Abdelshakour Abuzneid e Arafat AbuMallouh: *Deep and Machine Learning Approaches for Anomaly-Based Intrusion Detection of Imbalanced Network Traffic.* IEEE Sensors Letters, 3(1):1–4, jan 2019, ISSN 2475-1472. 9

[40] Yulianto, Arif, Parman Sukarno e Novian Anggis Suwastika: *Improving adaboost-based intrusion detection system (ids) performance on cic ids 2017 dataset.* Em *Journal of Physics: Conference Series*, volume 1192, página 012018. IOP Publishing, 2019. 9

[41] Aksu, Doğukan, Serpil Üstebay, Muhammed Ali Aydin e Tülin Atmaca: *Intrusion detection with comparative analysis of supervised learning techniques and fisher score feature selection algorithm.* Em *International Symposium on Computer and Information Sciences*, páginas 141–149. Springer, 2018. 9

[42] Li, Jiabin, Ming Liu, Zhi Xue, Xiaochen Fan e Xiangjian He: *Rtvd: A real-time volumetric detection scheme for ddos in the internet of things.* IEEE Access, 8:36191–36201, 2020. 9

[43] Jia, Yizhen, Fangtian Zhong, Arwa Alrawais, Bei Gong e Xiuzhen Cheng: *Flowguard: An intelligent edge defense mechanism against iot ddos attacks.* IEEE Internet of Things Journal, 2020. 9

[44] Novaes, Matheus P, Luiz F Carvalho, Jaime Lloret e Mario Lemes Proença: *Long short-term memory and fuzzy logic for anomaly detection and mitigation in software-defined network environment.* IEEE Access, 8:83765–83781, 2020. 9

[45] AlEroud, A. e G. Karabatis: *A contextual anomaly detection approach to discover zero-day attacks.* Em *2012 International Conference on Cyber Security*, páginas 40–45, 2012. 10

[46] 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, mar 2018, ISSN 00344885. 12, 17

[47] Morcos, Faruck, Andrea Pagnani, Bryan Lunt, Arianna Bertolino, Debora S. Marks, Chris Sander, Riccardo Zecchina, José N. Onuchic, Terence Hwa e Martin Weigt: *Direct-coupling analysis of residue coevolution captures native contacts across many*

*protein families*. Proceedings of the National Academy of Sciences of the United States of America, 108(49):E1293–E1301, dec 2011, ISSN 00278424. 12, 15, 16

[48] Wu, F. Y.: *The Potts model*. Reviews of Modern Physics, 54(1):235–268, jan 1982, ISSN 0034-6861. 12

[49] Jaynes, E. T.: *Information theory and statistical mechanics. II*. Physical Review, 108(2):171–190, may 1957, ISSN 0031899X. 14, 15

[50] Giraud, BG, John M Heumann e Alan S Lapedes: *Superadditive correlation*. Physical Review E, 59(5):4983, 1999. 16

[51] Georges, Antoine e Jonathan S Yedidia: *How to expand around mean-field theory using high-temperature expansions*. Journal of Physics A: Mathematical and General, 24(9):2173, 1991. 16

[52] Plonka, David: *Flowscan: A network traffic flow reporting and visualization tool*. Em *LISA*, páginas 305–317, 2000. 19

[53] Lashkari, Arash Habibi, Gerard Draper-Gil, Mohammad Saiful Islam Mamun e Ali A Ghorbani: *Characterization of tor traffic using time based features*. Em *ICISSP*, páginas 253–262, 2017. 22, 23

[54] Russell, Stuart e Peter Norvig: *Artificial intelligence: a modern approach*. 2002. 23, 24

[55] Panda, Mrutyunjaya e Manas Ranjan Patra: *Network intrusion detection using naive bayes*. International journal of computer science and network security, 7(12):258–263, 2007. 23

[56] Mukherjee, Saurabh e Neelam Sharma: *Intrusion detection using naive bayes classifier with feature reduction*. Procedia Technology, 4:119–128, 2012. 23

[57] Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot e E. Duchesnay: *Scikit-learn: Machine learning in Python*. Journal of Machine Learning Research, 12:2825–2830, 2011. 24, 25, 26

[58] Li, Wenchao, Ping Yi, Yue Wu, Li Pan e Jianhua Li: *A new intrusion detection system based on knn classification algorithm in wireless sensor network*. Journal of Electrical and Computer Engineering, 2014, 2014. 24

[59] Quinlan, J Ross: *Simplifying decision trees*. International journal of man-machine studies, 27(3):221–234, 1987. 24, 28

[60] 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. 24

[61] 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. 25

[62] Freund, Yoav e Robert E Schapire: *A decision-theoretic generalization of on-line learning and an application to boosting*. Journal of computer and system sciences, 55(1):119–139, 1997. 25

[63] Hastie, Trevor, Saharon Rosset, Ji Zhu e Hui Zou: *Multi-class adaboost*. Statistics and its Interface, 2(3):349–360, 2009. 25

[64] Cortes, Corinna e Vladimir Vapnik: *Support-vector networks*. Machine learning, 20(3):273–297, 1995. 25, 28

[65] Rosenblatt, Frank: *The perceptron: a probabilistic model for information storage and organization in the brain*. Psychological review, 65(6):386, 1958. 26

[66] Shenfield, Alex, David Day e Aladdin Ayesh: *Intelligent intrusion detection systems using artificial neural networks*. ICT Express, 4(2):95–99, 2018. 26

[67] Kingma, Diederik P e Jimmy Ba: *Adam: A method for stochastic optimization*. arXiv preprint arXiv:1412.6980, 2014. 26

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

[69] Swain, Philip H e Hans Hauska: *The decision tree classifier: Design and potential*. IEEE Transactions on Geoscience Electronics, 15(3):142–147, 1977. 28

[70] Freund, Yoav, Robert E Schapire *et al.*: *Experiments with a new boosting algorithm*. Em *icml*, volume 96, páginas 148–156. Citeseer, 1996. 28

[71] Ho, Tin Kam: *Random decision forests*. Em *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, páginas 278–282. IEEE, 1995. 28

[72] McCulloch, Warren S e Walter Pitts: *A logical calculus of the ideas immanent in nervous activity*. The bulletin of mathematical biophysics, 5(4):115–133, 1943. 28

[73] Lewis, David D: *Naive (bayes) at forty: The independence assumption in information retrieval*. Em *European conference on machine learning*, páginas 4–15. Springer, 1998. 28

[74] Japkowicz, Nathalie e Mohak Shah: *Evaluating learning algorithms: a classification perspective*. Cambridge University Press, 2011. 29

[75] Brzezinski, Dariusz e Jerzy Stefanowski: *Prequential auc: properties of the area under the roc curve for data streams with concept drift*. Knowledge and Information Systems, 52(2):531–562, 2017. 29

[76] Wu, Shaomin, Peter Flach e Cèsar Ferri: *An improved model selection heuristic for auc*. Em *European Conference on Machine Learning*, páginas 478–489. Springer, 2007. 29

# Apêndice A

# Artigo sumetido para a revista IEEE TNSM

# A new method for flow-based network intrusion detection using the inverse Potts model

Camila Pontes, Manuela Souza, João Gondim, Matt Bishop and Marcelo Marotta

***Abstract*—Network Intrusion Detection Systems (NIDS) play an important role as tools for identifying potential network threats. In the context of ever-increasing traffic volume on computer networks, flow-based NIDS arise as good solutions for real-time traffic classification. In recent years, different flow-based classifiers have been proposed using Machine Learning (ML) algorithms. Nevertheless, classical ML-based classifiers have some limitations. For instance, they require large amounts of labeled data for training, which might be difficult to obtain. Additionally, most ML-based classifiers are not capable of domain adaptation, *i.e.,* after being trained on an specific data distribution, they are not general enough to be applied to other related data distributions. And, finally, many of the models inferred by these algorithms are black boxes, which do not provide explainable results. To overcome these limitations, we propose a new algorithm, called Energy-based Flow Classifier (EFC). This anomaly-based classifier uses inverse statistics to infer a statistical model based on labeled benign examples. We show that EFC is capable of accurately performing binary flow classification and is more adaptable to different data distributions than classical ML-based classifiers. Given the positive results obtained on three different datasets (CIDDS-001, CICIDS17 and CICDDoS19), we consider EFC to be a promising algorithm to perform robust flow-based traffic classification.**

***Index Terms*—Flow-based Network Intrusion Detection, Anomaly-based Network Intrusion Detection, Network Flow Classification, Network Intrusion Detection Systems, Energy-based Flow Classifier, Inverse Potts Model, Domain Adaptation.**

## I. Introduction

SYMANTEC'S Internet Security Threat Report [1] points out a 56% increase in the number of web attacks in 2019. Network scans, denial of service, and brute force attacks are among the most common threats. Such malicious activities threaten individuals and collective organizations such as public health, financial, and government institutions. In this context, Network Intrusion Detection Systems (NIDSs) play an important role as tools for managing and identifying potential threats in the network [2].

There are two main approaches for NIDSs regarding the kind of data analyzed: packet-based and flow-based. In the former, deep packet inspection is performed, taking into account individual packet payloads and header information [3]. In the latter, flows, *i.e.,* packet collections, are analyzed regarding their properties, *e.g.,* duration, number of packets, number of bytes, and source/destination port [3]. To perform classification in real-time, a massive volume of data must be analyzed, making deep packet inspection too costly to be applied regarding processing and energy consumption. Since flow-based approaches can classify the whole traffic inspecting an equivalent to 0.1% of the total volume, NIDSs based on flow analysis arise as good solutions for real-time traffic classification [4]. Besides, with the advent of software-defined networking and the virtualization of network functions, distributed security systems can take advantage of the spread of NIDSs based on flow analysis to improve their security management across the network [5].

In recent years, different flow-based classifiers have been proposed based on both shallow and deep learning [6]. According to the report in [6], the best flow-based classifiers achieve around 99% accuracy. Although quite accurate, classical Machine Learning (ML)-based classifiers require labeled malicious traffic samples to perform training. However, real traffic labeling might be difficult, especially in the case of malicious traffic [7]. Besides, ML-based classifiers after trained on specific data distribution usually do not work well when applied to other data with slightly different distributions, *i.e.,* they have low domain adaptation capability [8], [9], [10]. Such a capability is particularly important to the network context since a standard procedure is to perform the training of classifiers with simulated data and, afterward, apply in real scenarios that change the data distribution requiring domain adaptation. Moreover, most ML algorithms are well-known to be black-box mechanisms, challenging to be understood and readjusted in detail [11], [12]. In this regard, there is a clear need for a new flow-based classifier for NIDSs, which generates an understandable model (white box) based solely on benign examples, and adaptable to different domains.

In this work, we propose a novel classifier called Energy-based Flow Classifier (EFC), which is inspired by the inverse Potts model from quantum mechanics and adapted to network flow classification. EFC performs one-class, anomaly-based classification, *i.e.,* as long as it can learn the properties of benign flows, it will discriminate between benign and malicious flows. Moreover, it is a white box algorithm, producing a statistical model that can be analyzed in detail regarding individual parameter values. Here, we compared the performance of EFC against a variety of classifiers using three

C. Pontes, M. Souza, J. Gondim and M. A. Marotta are with the University of Brasilia, Brazil, emails: cftpontes@gmail.com, gondim@unb.br, marcelo.marotta@unb.br;

M. Bishop is with the University of California at Davis, Davis, USA, email: mabishop@ucdavis.edu

different datasets, *i.e.,* CIDDS-001 [13], CICIDS17 [14], and CICDDoS19 [15]. Our results show that EFC's performance is comparable to the performance of the other classifiers. Also, we observed that EFC is less sensitive to changes in data distribution than the others. Our main contributions are:

- The proposal and implementation of a flow classifier based on the inverse Potts model to be employed in NIDSs;
- A performance comparison of the proposed classifier with classical ML-based classifiers using three different datasets;
- An analysis of how different classifiers perform when trained within one domain and tested in another related domain.

The rest of this paper is structured as follows. In Section II, we briefly present the state-of-the-art in flow-based NIDSs. In Section III, we describe the structure of network flows with a preliminary analysis of the datasets considered here. In Section IV, we introduce the statistical model proposed and the classifier implementation. In Section V, we present the results obtained regarding the statistical model's analysis and the classification experiments performed. Finally, in Section VI, we present our conclusions and future work.

## II. Related Work

In this section, we first briefly review the state-of-the-art in flow-based network intrusion detection systems. In the following, some previous work on CIDDS-001, CICIDS17, and CICDDoS19 are shown to highlight their relevance as up-to-date datasets to be used in our experiments. Finally, some challenges of ML-based traffic classification are regarded, such as the difficulty in obtaining sufficient labeled data, the non-interpretability of models, and the difficulty in adapting to different domains (data distributions).

Several ML-based classifiers have been explored over the last years for network intrusion detection. Vinayakumar *et al.* (2017) [16], Mahfouz *et al.* (2020) [17] and Khan *et al.* (2020) [18] independently evaluated the performance of different ML-based classifiers over internet traffic datasets. In [16], the KDDCup'99 and NSL-KDD datasets are regarded to evaluate the performance of both shallow and deep learning-based classifiers. It is shown that deep learning-based approaches performed better to differentiate malicious attacks from benign traffic. Meanwhile, the authors of [17] considered the NSL-KDD dataset to compare the performance of different shallow learning-based classifiers. The classifier that presented the best performance without feature selection was the Decision Tree (DT). With feature selection, K-Nearest Neighbors (KNN) performed better to classify malicious traffic. Finally, the authors of [18] compared the performance of a few different classifiers over the UNSW-NB15 dataset. They observed that Random Forest (RF) overperformed all other classifiers. In fact, RF has been used in several recent NIDSs [19], [20], [21]. All of the aforementioned works use the F1-score as a metric to assess the performance of the different classifiers. In the present work, we consider both deep and shallow learning-based classifiers as baselines to assess EFC's performance over three different datasets, regarding the F1-score as one of the evaluation metrics.

To assess EFC's performance, one of the datasets we use is CIDDS-001. This dataset was used by Verma and Ranga (2018) [22] to assess the performance of KNN and k-means clustering algorithms. Both algorithms achieved over 99% accuracy. Also, Ring *et al.* [23] explored slow port scans detection using CIDDS-001. The approach proposed by them is capable of accurately recognizing the attacks with a low false alarm rate. Finally, Abdulhammed *et al.* [24] also performed classification based on flows on CIDDS-001 and proposed an approach that is robust considering imbalanced network traffic. In summary, CIDDS-001 is an updated and relevant dataset to be used for network flow-classification solutions, being one of our dataset choices for assessing the performance of EFC.

The other two datasets considered here are CICIDS17 and CICDDoS19, from the Canadian Institute for Cyber Security. Recently, Yulianto, Sukarno, and Suwastika [25] used CICIDS17 to assess the performance of an Adaboost-based classifier. Aksu *et al.* [26] did the same in 2018 with different ML classifiers. CICIDS17 contains benign as well as the most up-to-date common attacks, resembling true real-world data, being a relevant dataset to consider for flow-based traffic classification. Meanwhile, CICDDoS19 is a very recent dataset with a focus on DDoS attacks. [27] (2020) proposes a real-time entropy-based NIDS for detection of volumetric DDoS in Internet of Things (IoT) and performs tests over CICDDoS19 dataset, among other datasets. Another recent work [28] obtained over 99% accuracy over CICDDoS19 dataset using a Convolutional Neural Network (CNN). And, finally, Novaes *et al.* [29] proposed a system for intrusion detection based on fuzzy logic, which had its performance assessed on CICDDoS19. The rising popularity of this dataset serves as evidence of its relevance to assess the performance of different NIDS. Hence, we use CICIDS17 and CICDDoS19 datasets to test our classifier and compare it to the performance of classical ML classifiers.

Umer et al. (2017) [6] performed a comprehensive literature survey on flow-based network intrusion detection. In their work, some disadvantages of using ML-based classifiers for traffic classification are mentioned. Among them are the high computational cost of training the classifiers, the difficulty in obtaining representative datasets, and the high false positive rates observed. The present work addresses some of the issues mentioned, since the classifier proposed here has a low computational cost and learns exclusively based on benign samples. In the following, these and some other issues are discussed in further detail.

One of the commonly discussed issues in the field of ML is the tight dependency most algorithms have on the amount

of labeled samples available for training [7], which might be difficult to obtain in some contexts. For instance, it is difficult to obtain malicious traffic samples and to label it in the real world and this is why most of the network intrusion detection datasets contain simulated attacks. This issue makes it difficult to train intrusion detection algorithms in such a way that they might be able to detect zero-day threats [7]. The only way of possibly detecting a zero-day attack is relying on an anomaly-based classifier [30], such as the one we propose in this work. EFC, has a great advantage over other ML-based algorithms, which is the capability to infer a model based solely on benign traffic samples, *i.e.,* half of the information. Such capability can be used to circumvent the problem of obtaining a high amount of data and the labeling of malicious samples.

Another common problem in ML is that inferred models lose their predictive performance when tested in different domains (data distributions) [10]. In the field of network security, this adaptability is specially important given the existence of zero-day threats and the artificiality of most datasets used for research. In [10], there is an interesting discussion about the existing differences between datasets used by academics to test NIDSs and the network traffic observed in the real world. Additionally, the work of Bartos *et al.* [8] and Li *et al.* [9] also address this issue. They propose similar approaches, applying data transformations to the data to reduce differences between data distributions in different domains. In our work, we propose a classifier that is intrinsically adaptable to different domains, since the model inference is based solely on benign samples. Therefore, there is no need to transform the data in order to adapt the model or perform adjustments to a different domain, making our approach simpler and more straightforward.

Finally, another big issue in ML is the non-interpretability of some models [11], [12]. Artificial Neural Networks (ANNs), in special, became more and more opaque with time, despite overperforming other approaches in many tasks. The authors of [12] highlight that the best ML algorithms are not interpretable, hence the decision taken by them can not be explained. However, different contexts require transparent decision making and that is why the development of explainable models is so important. The authors of [11] call attention to the fact that trying to explain black box models might not be the best approach to solve the issue of non-interpretability. Instead, it is suggested the design of new models that are inherently interpretable. In line with what has been suggested by these recent studies, EFC generates a white-box model and, therefore, satisfies the requirement of providing explainable results, allowing classification results to be analysed in retrospect if needed. Thus, next, we introduce main concepts and intuitions to serve as basis for EFC.

## III. Preliminaries

In this section, we present some fundamental concepts to understand flow-based network intrusion detection. First, the concept of network flow and its features are introduced. In the following, the three internet flow datasets used in this work are presented and described in detail to provide concrete examples of features and contextualize the experimental results presented in the Results section. The information provided in this section serves as a basis to understand how EFC works.

A network flow is a set of packets that traverses intermediary nodes between end-points within a given time interval. Under the perspective of an intermediary node, *i.e.,* an observation point, all packets belonging to a given flow have a set of common features called flow keys. It means that flow keys do not change for packets belonging to the same flow, while the remaining features might vary. FlowScan [31] is an example of a tool capable of collecting data from a set of packets and extracting flow features to be later exported in different formats, such as NetFlow and IPFIX. Since NetFlow is the most commonly used format, its main features are listed below:

- Source/Destination IP (flow keys) - determine the origin and destination of a given flow in the network;
- Source/Destination port (flow keys) - characterize different kinds of network services *e.g.,* ssh service uses port 22;
- Protocol (flow key) - characterizes flows regarding the transport protocol used *e.g.,* TCP, UDP, ICMP.
- Number of packets (feature) - total number of packets captured in a flow;
- Number of bytes (feature) - total number of bytes in a flow;
- Duration (feature) - total duration of a flow in seconds;
- Initial timestamp (feature) - system time when one flow started to be captured.

Other features such as TCP Flags and Type of Service might also be exported in some cases. The combination of different flow keys and features characterize one flow and determine its particular behavior.

Flow-based approaches are seen as suitable alternatives to precede packet inspection in real-time NIDSs. The idea is to deeply inspect only the packets belonging to flows considered to be suspicious by the flow-based classifier. A two-step approach would notably reduce the amount of data analyzed while maintaining a high classification accuracy [4]. In this work, we are only concerned with the first step, which is the flow classification. We evaluate the performance of our algorithm, the EFC, compared to other ML algorithms using three different datasets. We also evaluate the performance of the algorithms by training with data from part of the dataset and testing with other parts of it. Nonetheless, although both parts of the data come from the same dataset, their distributions are different to characterize domain adaptation. In the following, we briefly describe the datasets used for testing and characterize what constitutes a domain adaptation in each of them.

### A. CIDDS-001

CIDDS-001 [13] is a relatively recent dataset composed of a set of flow samples captured within a simulated OpenStack environment and another set of flow samples obtained from a real server. The former contains only simulated traffic, while the latter includes both real and simulated traffic. Each sample collected within these two environments has one of the labels described in Table I.

Table I
LABELS WITHIN CIDDS-001 DATASET

| Environment | Labels |
|---|---|
| OpenStack | normal, DoS, portScan, pingScan, bruteForce |
| External server | normal, DoS, bruteForce, unknown, suspicious |

Simulated benign flows are labeled as *normal*, while simulated malicious flows are labeled as *dos*, *portScan*, *pingScan* or *bruteForce*, depending on the type of attack simulated. The labels *suspicious* and *unknown*, in turn, are used for real traffic. The external server is open to user access through the ports 80 and 443. Hence, flows directed at these ports were labeled as *unknown*, since they could be either benign or malicious. All flows directed at other ports were labeled as *suspicious*. Traffic was sampled in both the simulated and the external server environment for a period of four weeks. Within this dataset, a change from the simulated data distribution to the external server data distribution is a domain change, requiring the classifiers to adapt.

CIDDS-001 dataset flow features are shown in Table II. All features were taken into account for characterization and classification except for *Src IP*, *Dest IP* and *Date first seen*. These exceptions are because the latter one is intrinsically not informative to differentiate flows, and the former two are made up in the context of the simulated network and might be confounding.

Table II
FEATURES WITHIN CIDDS-001 DATASET

| # | Name | Description |
|---|---|---|
| 1 | Src IP | Source IP Address |
| 2 | Src Port | Source Port |
| 3 | Dest IP | Destination IP Address |
| 4 | Dest Port | Destination Port |
| 5 | Proto | Transport Protocol (*e.g.,* ICMP, TCP, or UDP) |
| 6 | Date first seen | Start time flow first seen |
| 7 | Duration | Duration of the flow |
| 8 | Bytes | Number of transmitted bytes |
| 9 | Packets | Number of transmitted packets |
| 10 | Flags | OR concatenation of all TCP Flags |

### B. CICIDS17

CICIDS17 [14] dataset contains benign traffic and the most up-to-date common attacks, resembling real-world data. This dataset was built using the abstract behavior of 25 users based on the HTTP, HTTPS, FTP, SSH, and email protocols. The data was captured during one week in July 2017. The attacks implemented include Brute Force FTP, Brute Force SSH, DoS, Heartbleed, Web Attack, Infiltration, Botnet, and DDoS. They were executed both morning and afternoon on Tuesday, Wednesday, Thursday, and Friday (see Table III).

Table III
ATTACKS WITHIN CICIDS17 DATASET

| Week day | Attacks |
|---|---|
| Monday | |
| Tuesday | FTP-Patator, SSH-Patator |
| Wednesday | DoS slowloris, DoS Slowhttptes, DoS Hulk, DoS GoldenEye, Heartbleed Port 444 |
| Thursday | Brute Force, XSS, Sql Injection, Dropbox download, Cool disk |
| Friday | Botnet ARES, Port Scan, DDoS LOIT |

Flow features on this dataset were extracted using CICFlowMeter [32]. There are in total 88 features, which are not going to be cited here because of the limited space. All features were considered here, except for Flow ID, Source IP, Destination IP, and Timestamp. These exceptions were made because the features were either intrinsically not informative or made up within a simulated environment.

### C. CICDDoS19

CICDDoS19 [15] contains benign traffic and the most up-to-date common DDoS attacks (volumetric and application: low volume, slow rate), resembling real-world data. This dataset contains different modern reflective DDoS attacks such as PortMap, NetBIOS, LDAP, MSSQL, UDP, UDP-Lag, SYN, NTP, DNS, and SNMP. The traffic was captured in January (first day) and March (second day), 2019. Attacks were executed during this period (see Table IV).

Table IV
ATTACKS WITHIN CICDDOS19 DATASET

| Day | Attacks |
|---|---|
| First | PortMap, NetBIOS, LDAP, MSSQL, UDP, UDP-Lag, SYN |
| Second | NTP, DNS, LDAP, MSSQL, NetBIOS, SNMP, SSDP, UDP, UDP-Lag, WebDDos, SYN, TFTP |

Flow features on this dataset were extracted using CICFlowMeter [32]. All features were considered here, except for Flow Id, Source IP, Destination IP, and Timestamp. These exceptions were made because the features were either intrinsically not informative or made up within a simulated environment.

Considering each concept regarding network flows, their features, and how they are presented across different datasets, serve as basisto introduce the main intuition behind EFC, such as presented next.

## IV. Statistical model

EFC is based on inverse statistics. The main task of inverse statistics is to infer a statistical distribution based on a sample of it [33]. Methods using inverse statistics have been successfully applied to problems in other disciplines, *e.g.,* predicting protein contacts in Biophysics [33], [34]. Here, the statistical inference is based on the Potts model [35]. This model provides a mathematical description of interacting spins on a crystalline lattice. Within the model framework, interacting spins are mapped into a graph $G(\eta,\varepsilon)$ (see Figure 1 A)), where each node $i \in \eta = \{1,...,N\}$ has an associated spin $a_i$, which can assume one value from a set $\Omega$ that contains all possible individual quantum states. Each node $i$ has also an associated local field $h_i(a_i)$ that is a function of $a_i$'s state. Meanwhile, each edge $(i,j) \in \varepsilon$, $i,j \in \eta$, has an associated coupling value $e_{ij}(a_i,a_j)$ that is a function of the states of spins $a_i$ and $a_j$ associated to nodes $i$ and $j$. A specific system configuration has an associated total energy, determined by the Hamiltonian function $\mathscr{H}(a_1...a_N)$, which depends on all spin states.

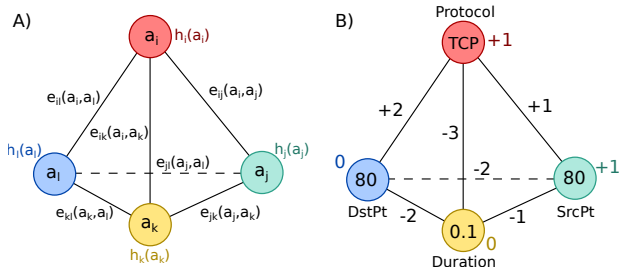


Figure 1. A) Interacting spins on a crystalline lattice. B) Network flow mapped into a graph structure.

In this work, we reuse the intuitions from the Potts model to characterize network flows (see Figure 1 B)). An individual flow $k$ is represented by a specific graph configuration $G_k(\eta,\varepsilon)$. Instead of spins, each node represents a selected feature $i \in \eta = \{SrcPort,...,Flags\}$. Within a given flow $k$, each feature $i$ assumes one value $a_{ki}$ from the set $\Omega_i$ that contains all possible values for this feature. As in the Potts Model, each feature $i$ has an associated local field $h_i(a_{ki})$. Meanwhile, $\varepsilon = \{(i,j)|i,j \in \eta; i \neq j\}$ is the set of edges determined by all possible pairs of features. Each edge has an associated coupling value determined by the function $e_{ij}(a_{ki},a_{kj})$.

Since the values of local fields and couplings depend on the values assumed by features within a given flow, each distinct flow will have a different combination of these quantities. As in the Potts Model, local fields and couplings determine the total "energy" $\mathscr{H}(a_{k1}...a_{kN})$ of each flow. For instance, in Figure 1 B), the total "energy" of the flow is obtained by summing up all values associated with the edges and to the nodes, resulting in a total of -3. Note that what we call energy is analogous to the notion of Hamiltonian in Quantum Mechanics. It is important to note that the model described here is discrete, therefore continuous features must be discretized. The classes for continuous feature discretization are shown in the Results section. In the following, we present the framework applied to perform the statistical model inference and subsequent energy-based flow classification.

### A. Model inference

In this section, a statistical model is going to be inferred in terms of couplings and local field values to perform energy-based flow classification. The main idea consists in extracting a statistical model from benign flow samples to infer coupling and local field values that characterize this type of traffic. When calculating the energies of unlabeled flows using the inferred values, it is expected that benign flows will have lower energies than malicious flows.

Let $(A_1...A_N)$ be an N-tuple of features, which can be instantiated for flow $k$ as $(a_{k1}...a_{kN})$, with $a_{k1} \in \Omega_1,...,a_{kN} \in \Omega_N$. Each feature value $a_{ki}$ is encoded by an integer from the set $\Omega = \{1,2,...,Q\}$, *i.e.,* all feature alphabets are the same $\Omega_i = \Omega$ of size $Q$. If a given feature can only assume $M$ values and $M < Q$, it is considered that values $M+1,...,Q$ are possible, but will never be observed empirically. For instance, if the only possible values for feature *protocol* are {*'TCP'*, *'UDP'*}, and given $Q = 4$. In this case, we would have the mapping {*'TCP'*:1, *'UDP'*:2, *' '*:3, *' '*:4 } and feature values 3 and 4 would never occur.

Now, let $\mathscr{K}$ be the set of all possible flows, *i.e.,* all possible combinations of feature values ($\mathscr{K} = \Omega^N$), and let $\mathscr{S} \subset \mathscr{K}$ be a sample of flows. We can use inverse statistical physics to infer a statistical model associating a probability $P(a_{k1}...a_{kN})$ to each flow $k \in \mathscr{K}$ based on sample $\mathscr{S}$. The global statistical model $P$ is inferred following the Entropy Maximization Principle [36]:

$$\max_{P} \quad -\sum_{k\in\mathscr{K}} P(a_{k1}...a_{kN}) log(P(a_{k1}...a_{kN})) \tag{1}$$

$$s.t.$$

$$\sum_{k\in\mathscr{K}|a_{ki}=a_i} P(a_{k1}...a_{kN}) = f_i(a_i) \tag{2}$$

$$\forall i \in \eta;\ \forall a_i \in \Omega;$$

$$\sum_{k\in\mathscr{K}|a_{ki}=a_i,a_{kj}=a_j} P(a_{k1}...a_{kN}) = f_{ij}(a_i,a_j) \tag{3}$$

$$\forall (i,j) \in \eta^2 \mid i \neq j;\ \forall (a_i,a_j) \in \Omega^2;$$

where $f_i(a_i)$ is the empirical frequency of value $a_i$ on feature $i$ and $f_{ij}(a_i,a_j)$ is the empirical joint frequency of the pair of values $(a_i,a_j)$ of features $i$ and $j$. Note that constraints 2 and 3 force model $P$ to generate single as well as joint empirical

frequency counts as marginals. This way, the model is sure to be coherent with empirical data.

Single and joint empirical frequencies $f_i(a_i)$ and $f_{ij}(a_i,a_j)$ are obtained from set $\mathscr{S}$ by counting occurrences of a given feature value $a_i$ or feature value pair $(a_i,a_j)$, respectively, and dividing by the total number of flows in $\mathscr{S}$. Since the set $\mathscr{S}$ is finite and much smaller than $\mathscr{K}$, inferences based on $\mathscr{S}$ are subjected to undersampling effects. Following the theoretical framework proposed in [34], we add pseudocounts to empirical frequencies to limit undersampling effects by performing the following operations:

$$f_i(a_i) \leftarrow (1-\alpha)f_i(a_i) + \frac{\alpha}{Q} \tag{4}$$

$$f_{ij}(a_i,a_j) \leftarrow (1-\alpha)f_{ij}(a_i,a_j) + \frac{\alpha}{Q^2} \tag{5}$$

where $(a_i,a_j) \in \Omega^2$ and $0 \leq \alpha \leq 1$ is a parameter defining the weight of the pseudocounts. The introduction of pseudocounts is equivalent to assuming that $\mathscr{S}$ is extended with a fraction of flows with uniformly sampled features.

The proposed maximization can be solved using a Lagrangian function such as presented in [36], yielding the following Boltzmann-like distribution:

$$P^*(a_{k1}...a_{kN}) = \frac{e^{-\mathscr{H}(a_{k1}...a_{kN})}}{Z} \tag{6}$$

where

$$\mathscr{H}(a_{k1}...a_{kN}) = -\sum_{i,j|i<j} e_{ij}(a_{ki},a_{kj}) - \sum_i h_i(a_{ki}) \tag{7}$$

is the Hamiltonian of flow $k$ and $Z$ (eq. (6)) is the partition function that normalizes the distribution. Since in this work we are not interested in obtaining individual flow probabilities, $Z$ is not required and, as a consequence, its calculation is omitted. Our objective is to calculate individual flows energies, *i.e.,* individual Hamiltonians as determined in eq. (7).

Note that the Hamiltonian, as presented above, is fully determined regarding the Lagrange multipliers $e_{ij}(\cdot)$ and $h_i(\cdot)$ associated to constraints (2) and (3), respectively. Within the Potts Model framework, the Lagrange multipliers have a special meaning, with the set $\{e_{ij}(a_i,a_j)|(a_i,a_j) \in \Omega^2\}$ being the set of all possible coupling values between features $i$ and $j$ and $\{h_i(a_i)|a_i \in \Omega\}$ the set of possible local fields associated to feature $i$.

Inferring the local fields and pairwise couplings is difficult since the number of parameters exceeds the number of independent constraints. Due to the physical properties of interacting spins, it is possible to infer pairwise coupling values $e_{ij}(a_i,a_j)$ using a Gaussian approximation. Assuming that the same properties apply for flow features, we infer coupling values as follows:

$$e_{ij}(a_i,a_j) = -(C^{-1})_{ij}(a_i,a_j), \tag{8}$$
$$\forall (i,j) \in \eta^2, \forall (a_i,a_j) \in \Omega^2, a_i,a_j \neq Q$$

where

$$C_{ij}(a_i,a_j) = f_{ij}(a_i,a_j) - f_i(a_i)f_j(a_j) \tag{9}$$

is the covariance matrix obtained from single and joint empirical frequencies. Taking the inverse of the covariance matrix is a well known procedure in statistics to remove the effect of indirect correlation in data [37]. Now, it is important to clarify that the number of independent constraints in eq. (2) and eq. (3) is actually $\frac{N(N-1)}{2}(Q-1)^2 + N(Q-1)$, even though the model in eq. (6) has $\frac{N(N-1)}{2}Q^2 + NQ$ parameters. So, without loss of generality, we set:

$$e_{i,j}(a_i,Q) = e_{i,j}(Q,a_j) = h_i(Q) = 0 \tag{10}$$

Thus, in eq. (8) there is no need to calculate $e_{i,j}(a_i,a_j)$in case $a_i$ or $a_j$ is equal to $Q$ [34]. Afterwards, local fields $h_i(a_i)$ can be inferred using a mean-field approximation [38]:

$$\frac{f_i(a_i)}{f_i(Q)} = exp\left(h_i(a_i) + \sum_{j,a_j} e_{ij}(a_i,a_j)f_j(a_j)\right), \tag{11}$$
$$\forall i \in \eta, a_i \in \Omega, a_i \neq Q$$

where $f_i(Q)$ is the frequency of the last element $a_i = Q$ for any feature $i$ used for normalization. It is also worth mentioning that the element Q is arbitrarily selected and could be replaced by any other value in $\{1...Q\}$ as long as the selected element is kept the same for calculations of the local fields of every feature $i \in \eta$. Note that in eq. (11) the empirical single frequencies $f_i(a_i)$ and the coupling values $e_{ij}(a_i,a_j)$ are known, yielding:

$$h_i(a_i) = ln\left(\frac{f_i(a_i)}{f_i(Q)}\right) - \sum_{j,a_j} e_{ij}(a_i,a_j)f_j(a_j) \tag{12}$$

In the mean-field approximation presented above, the interaction of a feature with its neighbors is replaced by an approximate interaction with an averaged feature, yielding an approximated value for the local field associated to it.

For further details about these calculations, please refer to [33]. Now that all model parameters are known, it is possible to calculate a given flow energy according to eq. (7). In the following, we are going to present the theoretical framework implementation to perform a two-class, *i.e.,* benign and malicious, flow classification, *i.e.,* Energy-based flow classification (EFC).

### B. Energy-based flow classification

The energy of a given flow can be calculated according to eq. (7) based on the values of its features and the parameters from the statistical model inferred in the last section. In simple terms, a given flow energy is the negative sum of couplings and local fields associated with its features, according to a given statistical model. It means that a flow that resembles the ones used to infer the model is likely to be low in energy.

Since EFC is an anomaly-based classifier, the statistical model used for classification is inferred based only on benign

flow samples. We would then expect the energies of benign samples to be lower than the energies of malicious samples. In this sense, it is possible to classify flow samples as benign or malicious based on a chosen energy threshold. The classification is performed by stating that samples with energy smaller than the threshold are benign, and samples with energy greater than or equal to the threshold are malicious. Note that the threshold for classification can be chosen in different ways, and it can be static or dynamic. In this work, we will consider a static threshold.

**Algorithm 1** Energy-based Flow Classifier

```
    Input: benign_flows_(K×N), Q, α, cutoff
 1: import all model inference functions
 2: f_i ← SiteFreq(benign_flows, Q, α)
 3: f_ij ← PairFreq(benign_flows, f_i, Q, α)
 4: e_ij ← Couplings(f_i, f_ij, Q)
 5: h_i ← LocalFields(e_ij, f_i, Q)
 6: while Scanning the Network do
 7:     flow ← wait_for_incoming_flow()
 8:     e ← 0
 9:     for i ← 1 to N − 1 do
10:         a_i ← flow[i]
11:         for j ← i + 1 to N do
12:             a_j ← flow[j]
13:             if a_i ≠ Q and a_j ≠ Q then
14:                 e ← e − e_ij[i, a_i, j, a_j]
15:             end if
16:         end for
17:         if a_i ≠ Q then
18:             e ← e − h_i[i, a_i]
19:         end if
20:     end for
21:     if e ≥ cutoff then
22:         stop_flow()
23:         forward_to_DPI()
24:     else
25:         release_flow()
26:     end if
27: end while
```

Algorithm 1 shows the implementation of EFC. In lines 2-5, the statistical model for the sampled flows is inferred, as described by eqs. (4), (5), (8) and (12). Afterward, on lines 6-27, the classifier monitors the network waiting for a captured flow. When a flow is captured, its energy is calculated on lines 9-20, according to the Hamiltonian in eq. (7). The computed flow energy is compared to a known threshold (*cutoff*) value on line 21. In case the energy falls above the threshold, the flow is classified as malicious and should be forwarded to deep packet inspection (line 23) for assessment. Otherwise, the flow is released, and the classifier waits for another flow.

It is essential to highlight that the time complexity of the training step of EFC is $O((M \times Q)^3 + N \times M^2 \times Q^2)$, where $N$ is the number of samples, $M$ is the number of features, and $Q$ is the size of the alphabet. Meanwhile, the complexity of the classification step for each sample is $O(M)$. It means that, in both steps, the complexity is more dependant on the number of features chosen, which can be kept small by using a feature selection mechanism, *e.g.,* Principal Component Analysis (PCA). Therefore, it is possible to see that EFC has a low computational cost when compared to ML-based classifiers, such as ANN, Support Vector Machine (SVM), and RF. Considering the implementation shown in this section, next, we present the results obtained when EFC is used to perform flow classification.

## V. Results

In this section, we present the results obtained for EFC and classical ML-based classifiers in different binary classification experiments considering three different datasets, *i.e.,* CIDDS-001, CICIDS17, and CICDDoS19. First, we show that EFC can separate benign from malicious flows based on their energies, a result that is consistent for all the considered datasets. Then, we present EFC's classification performance and compare it to the classification performance of classical ML-based classifiers in different experiments.

It is important to highlight that the classification experiments we perform in this work were designed not only to assess the performance of different classifiers but also to investigate their capability of adaptation to different domains, *i.e.,* data distributions. Hence, we performed two kinds of experiments: training/testing in the same domain, and training/testing in different domains. For training/testing in the same domain, in each experiment, we assessed the average performance of the classifiers over ten different test sets, containing 10,000 benign and 10,000 malicious samples each, randomly selected from the full dataset. Models were inferred based on 80% the test set and tested on the remaining 20%. The inferred models were used for each experiment to assess the performance of the classifiers over another ten test sets composed of 2,000 benign and 2,000 malicious samples from another domain (data distribution).

### A. EFC characterization

To assess the EFC classification capabilities to separate benign from malicious traffic flow samples correctly, we inferred a model based on benign samples from the OpenStack (simulated) environment within the CIDDS-001 dataset. This model was used to calculate the energy of benign and malicious flow samples coming from the simulated traffic, and also from the external server traffic. Figure 2A shows energy values of 40,000 classified flow samples, a merge of the results obtained over ten randomly sampled test sets. The statistical model used to calculate the energies in each test set was inferred based on 8,000 benign flows randomly sampled from the simulated traffic. The separation between the two flow classes is clear, *i.e.,* benign flows energy distribution is clearly shifted to the left in relation to malicious flows energy distribution.

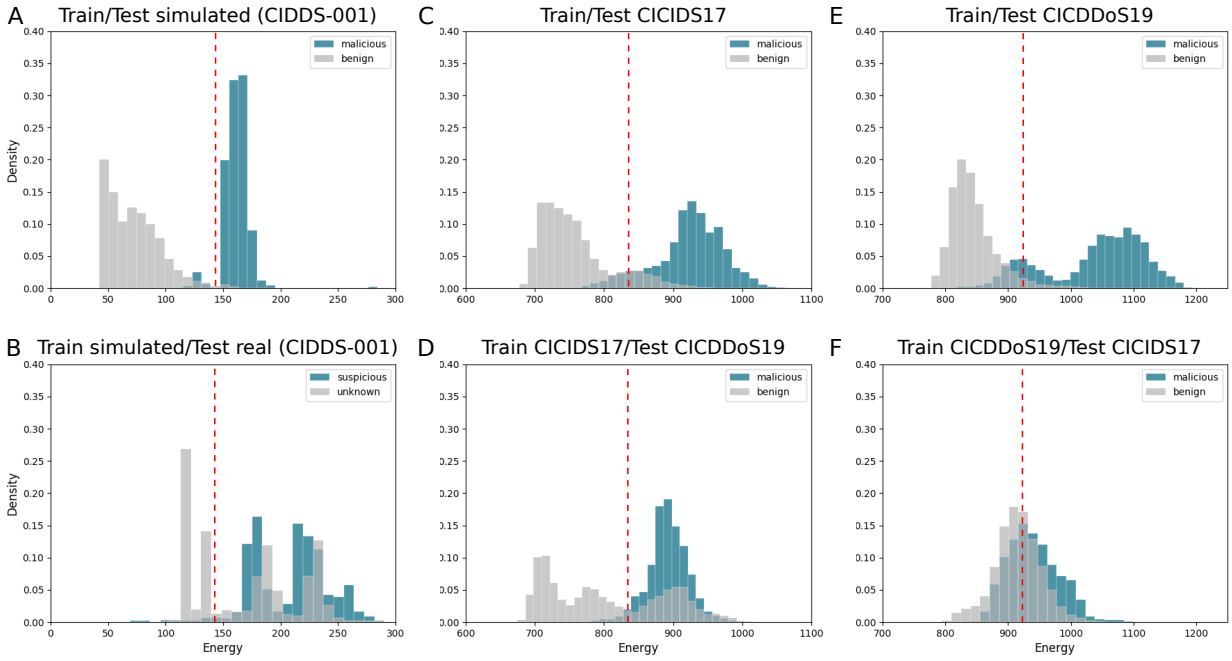


Figure 2. Energy histograms of benign (n = 20,000 in each plot) and malicious (n = 20,000 in each plot) flow samples obtained in the testing phase of a classification experiment performed over CIDDS-001 dataset (A,B), and two crossdataset classification experiments performed over CICIDS17 and CICDDoS19 dataset (C-F).

From the real traffic (captured in an external server) in CIDDS-001, energy values of 20,000 randomly sampled flows labeled as *unknown* and 20,000 randomly sampled flows labeled as *suspicious* are shown in Figure 2B. According to the data set definitions, traffic labeled as *unknown* is traffic at expected open ports of the server (port 80 and port 443). In this sense, this traffic can be either benign or malicious. Traffic labeled as *suspicious*, on the other hand, is unexpected traffic aimed at ports other than the expected ones. Hence, this traffic is considered malicious traffic. Note that the separation between these two classes is not so evident, since part of the *unknown* traffic mixes with *suspicious* traffic. However, it is possible to see that there is a distinct portion of the *unknown* traffic with lower energy than the rest and it can be distinctly noted. It is important to note that it is possible to apply the same energy threshold (red dashed line) to separate the two classes in both the simulated and the real environments. Because the threshold learned in the simulated environment can be applied to separate the traffic in the real environment, we understand that the statistical model learned is adaptable to different domains.

Figure 2C-F shows the results of a cross-dataset analysis performed on the remaining two datasets. When trained on CICIDS17, EFC is capable of clearly separating classes in this dataset (Figure 2C), as well as in CICDDoS19 (Figure 2D). Moreover, we can see that it is possible to find a unique threshold able to differentiate malicious from benign traffic, which reinforces the claim that EFC is adaptable to different domains. However, when trained on CICDDoS19, EFC is capable of distinguishing between classes in this dataset (Figure 2E), but its adaptation to CICIDS17 is not as good as it was for the previous case (Figure 2F). This is possibly due to the fact that CICDDoS19 attack classes are much more specific, restricted to DDoS, than those in CICIDS17, which covers a wider range of attack classes. However, even if the distributions overlap, it is still possible to learn a threshold that is acceptable for the classification for both cases, as indicated by the red dashed line.

In summary, the results presented in this subsection show that EFC can correctly discriminate between the two flow classes considered, *i.e.,* benign and malicious, and the results are consistent for all datasets considered. In addition to that, we observe that in all cases it is possible to find a classification threshold that can be applied to a different data distribution or domain. Such an observation contributes to the claim that EFC is adaptable to different domains and does not overfit data. In the following subsection, classification results are shown for different classifiers and compared with the results obtained for EFC.

### *B. Comparative analysis of EFC's performance*

We compared EFC to five different ML classifiers: K-Nearest Neighbors (KNN) [39], Decision Tree (DT) [40], [41], Multilayer perceptron (MLP) [42], Naive Bayes (NB) [43], and Support Vector Machine (SVM) [44], all deployed with their default scikit-learn[1] configurations. It is worthy mentioning

[1]Scikit-learn library - https://scikit-learn.org

that no ensemble methods were considered, since they present almost the same performance of the original method, *e.g.,* random forest and adaboost are derivative methods of DT and present almost the same performance when evaluated. Flow features were only discretized for EFC (Table V) since discretization would impair the performance of most ML algorithms. The metrics used to compare the results were the F1 score and the area under the ROC curve (AUC). The first metric, F1 score, is the harmonic mean of the Precision and the Recall, *i.e.,*

$$F1 = \frac{2}{Precision^{-1} + Recall^{-1}} = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \quad (13)$$

where $Precision = TP/(TP+FP)$, $Recall = TP/(TP+FN)$, TP are the true positives, *i.e.,* malicious traffic classified as malicious, FP are the false positives, *i.e.,* benign traffic classified as malicious, and FN are the false negatives, *i.e.,* malicious traffic classified as benign. The second metric, the area under the ROC curve (AUC), is one of the most widespread evaluation metrics for binary classifiers [45], [46]. The ROC curve is constructed by plotting the true positive rate (TPR) against the false positive rate (FPR) at different classification thresholds. It means that the AUC is the probability that a randomly chosen positive example will receive a higher score than a randomly chosen negative one. One of the main advantages of the AUC is that it is invariant to changes in class distribution, *i.e.,* the ROC curve will not change if the distribution changes in a test set, but the underlying conditional distributions from which the data are drawn stay the same [47], [46]. Since we are interested in evaluating domain adaptation, this metric is particularly interesting to be adopted in this work.

Table V shows the classes considered for feature discretization on CIDDS-001 dataset. Since *TCP Flags* is the discrete feature with more possible values (32 possibilities), the alphabet size $Q$ was set to 32. Each continuous feature values were clustered in a certain number of classes (or bins), up to $Q$ classes. Classes were determined in such a way that the number of values within each class was similar for all classes. Features within datasets CICIDS17 and CICDDoS19 were also discretized in such a way that the number of values within each bin was similar for all bins. These discretizations are not shown here because of the high number of features (>80 features) in these datasets.

Table V
CLASSES CONSIDERED FOR FEATURE DISCRETIZATION ON CIDDS-001

| Feature | List of classes upper limits |
|---|---|
| Duration | 0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.01, 0.04, 1, 10, 100, ∞ |
| Protocol | TCP, UDP, GRE, ICMP, IGMP |
| Src Port | 50, 60, 100, 400, 500, 40000, 60000, ∞ |
| Dst Port | 50, 60, 100, 400, 500, 40000, 60000, ∞ |
| Num. of Bytes | 50, 60, 70, 90, 100, 110, 200, 300, 400, 500, 700, 1000, 5000, ∞ |
| Num. of Packets | 2, 3, 4, 5, 6, 7, 10, 20, ∞ |
| TCP Flags | $\{(f_1, f_2, f_3, f_4, f_5) \mid f_i \in \{0,1\}\}$ |

To evaluate EFC's performance compared to other classifiers, we performed three independent experiments. The first experiment was performed on CIDDS-001. Training was performed on simulated flow samples, while testing was performed on both simulated and real flow samples captured in an external server. The second and the third experiments were cross-dataset experiments performed on CICIDS17 and CICDDoS19. In the former, training was performed on CICIDS17, with testing on both datasets, while in the latter training was performed on CICDDoS19, with testing on both datasets.

Table VI
AVERAGE COMPOSITION OF EACH OF THE TEST SETS IN EXPERIMENT 1

| CIDDS-001 OpenStack | | CIDDS-001 real traffic | |
|---|---|---|---|
| Label | Number | Label | Number |
| *normal* | 10,000 | *unknown* | 2,000 |
| *dos* | 9,800 | *suspicious* | 2,000 |
| *pingScan* | 20 | | |
| *portScan* | 150 | | |
| *bruteForce* | 30 | | |
| **Total** | 20,000 | **Total** | 4,000 |

Table VII
AVERAGE COMPOSITION OF EACH OF THE TEST SETS IN CROSS-DATASET EXPERIMENTS 2 AND 3

| CICIDS17 | | CICDDoS19 | |
|---|---|---|---|
| Label | Number | Label | Number |
| *benign* | 10,000 | *benign* | 10,000 |
| *FTP Patator* | 170 | *DrDoS DNS* | 890 |
| *SSH Patator* | 80 | *DrDoS LDAP* | 370 |
| *DDoS* | 2,740 | *DrDoS MSSQL* | 890 |
| *PortScan* | 1,060 | *DrDoS NetBIOS* | 880 |
| *Bot* | 110 | *DrDoS NTP* | 890 |
| *Infiltration* | 20 | *DrDoS SNMP* | 200 |
| *Brute force* | 50 | *DrDoS SSDP* | 890 |
| *SQL injection* | 10 | *DrDoS UDP* | 890 |
| *XSS* | 10 | *Syn* | 890 |
| *DoS Hulk* | 2,730 | *TFTP* | 890 |
| *DoS GoldenEye* | 2,730 | *LDAP* | 120 |
| *DoS Slowloris* | 120 | *NetBIOS* | 140 |
| *DoS Slowhttptest* | 170 | *MSSQL* | 660 |
| | | *Portmap* | 400 |
| | | *UDP* | 880 |
| | | *UDPLag* | 120 |
| **Total** | 20,000 | **Total** | 20,000 |

Essentially, in each experiment, we measured the performance of the classifiers when trained and tested in the same domain and when trained in one domain and tested in a different one. The performance was measured as the average over ten different test sets, composed of 10,000 benign and 10,000 malicious samples each, randomly selected form the

full dataset. With 80% of each test set being used for training and 20% for testing. The test sets containing samples from a different domain were not used for training, hence they were composed of only 2,000 benign and 2,000 malicious samples, randomly selected from the full dataset. EFC's cutoff was defined to be at the 95th percentile of the energy distribution obtained in the training phase **based solely in benign samples**. It means that we used a completely statistical threshold based only in the training considering benign traffic without adjustments based on malicious samples, such as other ML-Algorithms require. The average composition of the test sets is shown in Table VI and VII.

Table VIII shows the average performance and standard error (95% confidence interval) of each classifier on the first experiment considering CIDDS-001 dataset. When trained and tested in the same simulated environment, DT is the algorithm presenting the best performance, with an F1-score of $0.999 \pm 0.000$ and $0.999 \pm 0.000$ AUC. EFC does also perform well, being the second best in terms of AUC ($0.997 \pm 0.001$). When trained in the simulated environment and tested in the real environment, EFC outperforms the other classifiers in F1-score ($0.675 \pm 0.009$) and AUC ($0.720 \pm 0.001$). It is noteworthy that all algorithms present a considerable degradation in performance when tested in a different domain, showing how sensitive the inferred models are to changes in data distribution.

Table VIII
AVERAGE CLASSIFICATION PERFORMANCE AND STANDARD ERROR (95% CI) - TRAINING PERFORMED ON CIDDS-001 SIMULATED TRAFFIC

| | Train/Test simulated | | Train simulated/Test real | |
|---|---|---|---|---|
| Classifier | F1 score | AUC | F1 score | AUC |
| NB | 0.043 ± 0.016 | 0.502 ± 0.004 | 0.057 ± 0.024 | 0.517 ± 0.016 |
| KNN | 0.988 ± 0.001 | 0.994 ± 0.000 | 0.118 ± 0.014 | 0.524 ± 0.001 |
| DT | **0.999 ± 0.000** | **0.999 ± 0.000** | 0.556 ± 0.007 | 0.619 ± 0.000 |
| SVM | 0.805 ± 0.003 | 0.951 ± 0.002 | 0.531 ± 0.005 | 0.707 ± 0.003 |
| MLP | 0.979 ± 0.002 | 0.993 ± 0.001 | 0.151 ± 0.016 | 0.596 ± 0.002 |
| **EFC** | 0.975 ± 0.001 | 0.997 ± 0.001 | **0.675 ± 0.009** | **0.720 ± 0.001** |

Table IX shows the results of experiment two, which was performed on CICIDS17 and CICDDoS19 datasets. When trained and tested on CICIDS17, DT is again the algorithm to present the best performance both in terms of F1-score ($0.994 \pm 0.001$) and AUC ($0.994 \pm 0.001$), though indistinguishable from MLP AUC ($0.993 \pm 0.001$). Notably, when trained on CICIDS17 and tested on CICDDoS19, EFC outperformed the other algorithms in both F1-score ($0.787 \pm 0.004$) and AUC ($0.781 \pm 0.003$). Again, it is possible to see that EFC is the best algorithm in adapting to a different data distribution when evaluating both metrics.

Further, Table X shows the results of experiment three, which was performed on CICIDS17 and CICDDoS19 datasets. Once more, DT overperformed the other classifiers when training and testing on the same dataset, with F1-score of $0.998 \pm 0.000$ and AUC of $0.998 \pm 0.000$. When tested on the CICDDoS19 dataset though, EFC achieved the best F1-score ($0.641 \pm 0.002$), while KNN was the best in terms of AUC ($0.670 \pm 0.002$). EFC's AUC ($0.664 \pm 0.002$) was the second best, which means that EFC performance was good, taking into consideration both the F1-score and the AUC. Even though this adaptation seems more challenging than the previous ones, EFC's performance was consistent on all the experiments performed.

Table IX
AVERAGE CLASSIFICATION PERFORMANCE AND STANDARD ERROR (95% CI) - TRAINING PERFORMED ON CICIDS17

| | Train/Test CICIDS17 | | Train CICIDS17/Test CICDDoS19 | |
|---|---|---|---|---|
| Classifier | F1 score | AUC | F1 score | AUC |
| NB | 0.344 ± 0.049 | 0.413 ± 0.021 | 0.344 ± 0.049 | 0.413 ± 0.049 |
| KNN | 0.961 ± 0.001 | 0.987 ± 0.001 | 0.457 ± 0.046 | 0.771 ± 0.001 |
| DT | **0.994 ± 0.001** | **0.994 ± 0.001** | 0.168 ± 0.090 | 0.525 ± 0.001 |
| SVM | 0.930 ± 0.003 | 0.974 ± 0.001 | 0.264 ± 0.025 | 0.664 ± 0.003 |
| MLP | 0.961 ± 0.003 | **0.993 ± 0.001** | 0.221 ± 0.033 | 0.775 ± 0.003 |
| **EFC** | 0.898 ± 0.003 | 0.975 ± 0.001 | **0.787 ± 0.004** | **0.781 ± 0.003** |

Table X
AVERAGE CLASSIFICATION PERFORMANCE AND STANDARD ERROR (95% CI) - TRAINING PERFORMED ON CICDDOS19

| | Train/Test CICDDoS19 | | Train CICDDoS19/Test CICIDS17 | |
|---|---|---|---|---|
| Classifier | F1 score | AUC | F1 score | AUC |
| NB | 0.590 ± 0.006 | 0.428 ± 0.007 | 0.590 ± 0.006 | 0.428 ± 0.006 |
| KNN | 0.960 ± 0.002 | 0.984 ± 0.001 | 0.397 ± 0.043 | **0.670 ± 0.002** |
| DT | **0.998 ± 0.000** | **0.998 ± 0.000** | 0.259 ± 0.012 | 0.476 ± 0.000 |
| SVM | 0.933 ± 0.002 | 0.976 ± 0.002 | 0.239 ± 0.009 | 0.538 ± 0.002 |
| MLP | 0.968 ± 0.002 | 0.993 ± 0.001 | 0.227 ± 0.011 | 0.451 ± 0.002 |
| **EFC** | 0.916 ± 0.002 | 0.981 ± 0.001 | **0.641 ± 0.002** | 0.664 ± 0.002 |

Taken as a whole, the results presented in this subsection show that EFC is better at adapting to other domains than classical ML-based classifiers on average. In addition to that, it is possible to see that EFC achieves AUC values similar to the best ML algorithms when trained and tested in the same domain, showing that it is capable of performing well even if trained with only half of the information (benign data only) when compared to other classifiers (using malicious and benign data). Not using malicious samples in the training phase is likely to be the reason why EFC is so good at adapting to other domains. EFC's increased capability for domain adaptation when there is a significant difference in data distribution is a highly desirable trait in network flow-based classifiers, since changes in traffic composition are expected to be very frequent, and new kinds of attacks are generated continuously. In the next section, we present our conclusions and future work directions.

## VI. CONCLUSION

In this work, we present a new flow-based classifier for network intrusion detection called Energy-based Flow Classifier (EFC). In the training phase, EFC infers a statistical

model based solely on benign traffic samples. Afterward, this statistical model is used to classify network flows in benign or malicious based on "energy" values. Our results show that EFC is capable of correctly performing network flow binary classification considering three different datasets. F1 score (around 97% at best) and AUC (around 99% at best) values obtained using EFC are comparable to the values obtained with other classical ML-based classifiers, such as k-nearest neighbors, decision tree and multilayer perceptron, even though EFC uses only half of the information in the training phase compared to the other algorithms.

In addition to that, we analyzed different classifiers in terms of their capability for domain adaptation and observed that EFC is more suitable to that than classical ML-based algorithms. In all the experiments performed to evaluate that over different datasets, EFC outperformed the other classifiers in F1-score and was among the best ones in AUC. We understand that EFC's capability for domain adaptation is probably linked to the fact that the model inference is based only on benign samples, which helps preventing overfit.

Considering the advantages presented, we believe EFC to be a promising algorithm to perform flow-based traffic classification. Nevertheless, despite the promising results achieved, there is still room for further testing and improvement. For instance, to obtain a more homogeneous distribution of different attack types, we performed a dataset undersampling, which might have had some effect on the results. Hence, in future work, we aim at performing a more comprehensive investigation of EFC's applicability to real-world data and different contexts, such as fraud analysis in bank data.

## Acknowledgment

The authors would like to thank Luís Paulo Faina Garcia for helping with dataset analysis. Matt Bishop was supported by the National Science Foundation under Grant Number OAC-1739025. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. João Gondim gratefully acknowledges the support from Project "EAGER: USBRCCR: Collaborative: Securing Networks in the Programmable Data Plane Era" funded by NSF (National Science Foundation) and RNP (Brazilian National Research Network).

## References

[1] Symantec, "Internet Security Threat Report (ISTR) 2019 | Symantec," Apr. 2019. [Online]. Available: https://www.symantec.com/security-center/threat-report

[2] M. Pradhan, C. K. Nayak, and S. K. Pradhan, "Intrusion detection system (ids) and their types," in *Securing the Internet of Things: Concepts, Methodologies, Tools, and Applications*. IGI Global, 2020, pp. 481–497.

[3] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho, "A survey of network-based intrusion detection data sets," *Computers & Security*, 2019.

[4] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller, "An overview of IP flow-based intrusion detection," *IEEE Communications Surveys and Tutorials*, vol. 12, no. 3, pp. 343–356, 2010.

[5] J. C. Correa Chica, J. C. Imbachi, and J. F. Botero Vega, "Security in sdn: A comprehensive survey," *Journal of Network and Computer Applications*, p. 102595, 2020.

[6] M. F. Umer, M. Sher, and Y. Bi, "Flow-based intrusion detection: Techniques and challenges," *Computers and Security*, vol. 70, pp. 238–254, sep 2017.

[7] A. Singla, E. Bertino, and D. Verma, "Overcoming the lack of labeled data: training intrusion detection models using transfer learning," in *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 2019, pp. 69–74.

[8] K. Bartos, M. Sofka, and V. Franc, "Optimized invariant representation of network traffic for detecting unseen malware variants," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 807–822.

[9] H. Li, Z. Chen, R. Spolaor, Q. Yan, C. Zhao, and B. Yang, "Dart: Detecting unseen malware variants using adaptation regularization transfer learning," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–6.

[10] M. Zolanvari, M. A. Teixeira, and R. Jain, "Effect of imbalanced datasets on security of industrial iot using machine learning," in *2018 IEEE International Conference on Intelligence and Security Informatics (ISI)*. IEEE, 2018, pp. 112–117.

[11] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nature Machine Intelligence*, vol. 1, no. 5, pp. 206–215, 2019.

[12] A. Holzinger, "From machine learning to explainable ai," in *2018 World Symposium on Digital Intelligence for Systems and Machines (DISA)*. IEEE, 2018, pp. 55–66.

[13] M. Ring, S. Wunderlich, D. Grüdl, D. Landes, and A. Hotho, "Flow-based benchmark data sets for intrusion detection," in *Proceedings of the 16th European Conference on Cyber Warfare and Security. ACPI*, 2017, pp. 361–369.

[14] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization." in *ICISSP*, 2018, pp. 108–116.

[15] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, "Developing realistic distributed denial of service (ddos) attack dataset and taxonomy," in *2019 International Carnahan Conference on Security Technology (ICCST)*. IEEE, 2019, pp. 1–8.

[16] R. Vinayakumar, K. Soman, and P. Poornachandran, "Evaluating effectiveness of shallow and deep networks to intrusion detection system," in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 2017, pp. 1282–1289.

[17] A. M. Mahfouz, D. Venugopal, and S. G. Shiva, "Comparative analysis of ml classifiers for network intrusion detection," in *Fourth International Congress on Information and Communication Technology*. Springer, 2020, pp. 193–207.

[18] S. Khan, E. Sivaraman, and P. B. Honnavalli, "Performance evaluation of advanced machine learning algorithms for network intrusion detection system," in *Proceedings of International Conference on IoT Inclusive Life (ICIIL 2019), NITTTR Chandigarh, India*. Springer, 2020, pp. 51–59.

[19] X. Tan, S. Su, Z. Huang, X. Guo, Z. Zuo, X. Sun, and L. Li, "Wireless sensor networks intrusion detection based on smote and the random forest algorithm," *Sensors*, vol. 19, no. 1, p. 203, 2019.

[20] J. Kazemitabar, R. TAHERI, and G. KHERADMANDIAN, "A novel technique for improvement of intrusion detection via combining random forrest and genetic algorithm," 2019.

[21] T. T. Bhavani, M. K. Rao, and A. M. Reddy, "Network intrusion detection system using random forest and decision tree machine learning techniques," in *First International Conference on Sustainable Technologies for Computational Intelligence*. Springer, 2020, pp. 637–643.

[22] A. Verma and V. Ranga, "Statistical analysis of cidds-001 dataset for network intrusion detection systems using distance-based machine learning," *Procedia Computer Science*, vol. 125, pp. 709–716, 2018.

[23] M. Ring, D. Landes, and A. Hotho, "Detection of slow port scans in flow-based network traffic," *PloS one*, vol. 13, no. 9, p. e0204507, 2018.

[24] R. Abdulhammed, M. Faezipour, A. Abuzneid, and A. AbuMallouh, "Deep and Machine Learning Approaches for Anomaly-Based Intrusion Detection of Imbalanced Network Traffic," *IEEE Sensors Letters*, vol. 3, no. 1, pp. 1–4, jan 2019.

[25] A. Yulianto, P. Sukarno, and N. A. Suwastika, "Improving adaboost-based intrusion detection system (ids) performance on cic ids 2017 dataset," in *Journal of Physics: Conference Series*, vol. 1192, no. 1. IOP Publishing, 2019, p. 012018.

[26] D. Aksu, S. Üstebay, M. A. Aydin, and T. Atmaca, "Intrusion detection with comparative analysis of supervised learning techniques and fisher score feature selection algorithm," in *International Symposium on Computer and Information Sciences*. Springer, 2018, pp. 141–149.

[27] J. Li, M. Liu, Z. Xue, X. Fan, and X. He, "Rtvd: A real-time volumetric detection scheme for ddos in the internet of things," *IEEE Access*, vol. 8, pp. 36 191–36 201, 2020.

[28] Y. Jia, F. Zhong, A. Alrawais, B. Gong, and X. Cheng, "Flowguard: An intelligent edge defense mechanism against iot ddos attacks," *IEEE Internet of Things Journal*, 2020.

[29] M. P. Novaes, L. F. Carvalho, J. Lloret, and M. L. Proença, "Long short-term memory and fuzzy logic for anomaly detection and mitigation in software-defined network environment," *IEEE Access*, vol. 8, pp. 83 765–83 781, 2020.

[30] A. AlEroud and G. Karabatis, "A contextual anomaly detection approach to discover zero-day attacks," in *2012 International Conference on Cyber Security*, 2012, pp. 40–45.

[31] D. Plonka, "Flowscan: A network traffic flow reporting and visualization tool." in *LISA*, 2000, pp. 305–317.

[32] A. H. Lashkari, G. Draper-Gil, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of tor traffic using time based features." in *ICISSP*, 2017, pp. 253–262.

[33] S. Cocco, C. Feinauer, M. Figliuzzi, R. Monasson, and M. Weigt, "Inverse statistical physics of protein sequences: A key issues review," *Reports on Progress in Physics*, vol. 81, no. 3, p. 032601, mar 2018.

[34] F. Morcos, A. Pagnani, B. Lunt, A. Bertolino, D. S. Marks, C. Sander, R. Zecchina, J. N. Onuchic, T. Hwa, and M. Weigt, "Direct-coupling analysis of residue coevolution captures native contacts across many protein families," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 108, no. 49, pp. E1293–E1301, dec 2011.

[35] F. Y. Wu, "The Potts model," *Reviews of Modern Physics*, vol. 54, no. 1, pp. 235–268, jan 1982.

[36] E. T. Jaynes, "Information theory and statistical mechanics. II," *Physical Review*, vol. 108, no. 2, pp. 171–190, may 1957.

[37] B. Giraud, J. M. Heumann, and A. S. Lapedes, "Superadditive correlation," *Physical Review E*, vol. 59, no. 5, p. 4983, 1999.

[38] A. Georges and J. S. Yedidia, "How to expand around mean-field theory using high-temperature expansions," *Journal of Physics A: Mathematical and General*, vol. 24, no. 9, p. 2173, 1991.

[39] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[40] J. R. Quinlan, "Simplifying decision trees," *International journal of man-machine studies*, vol. 27, no. 3, pp. 221–234, 1987.

[41] P. H. Swain and H. Hauska, "The decision tree classifier: Design and potential," *IEEE Transactions on Geoscience Electronics*, vol. 15, no. 3, pp. 142–147, 1977.

[42] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[43] D. D. Lewis, "Naive (bayes) at forty: The independence assumption in information retrieval," in *European conference on machine learning*. Springer, 1998, pp. 4–15.

[44] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[45] N. Japkowicz and M. Shah, *Evaluating learning algorithms: a classification perspective*. Cambridge University Press, 2011.

[46] D. Brzezinski and J. Stefanowski, "Prequential auc: properties of the area under the roc curve for data streams with concept drift," *Knowledge and Information Systems*, vol. 52, no. 2, pp. 531–562, 2017.

[47] S. Wu, P. Flach, and C. Ferri, "An improved model selection heuristic for auc," in *European Conference on Machine Learning*. Springer, 2007, pp. 478–489.


**Camila F. T. Pontes** is a student at the University of Brasilia (UnB), Brasilia, DF, Brazil. She received her M.Sc. degree in Molecular Biology in 2016 from UnB and is currently an undergrad student at the Department of Computer Science (CIC/UnB). Her research interests are Computational and Theoretical Biology and Network Security.

**Manuela M. C. de Souza** is an undergrad Computer Science student at University of Brasilia (UnB), Brasilia, DF, Brazil. Her research interest is Network Security.

**João J. C. Gondim** was awarded an M.Sc. in Computing Science at Imperial College, University of London, in 1987 and a Ph.D. in Electrical Engineering at UnB (University of Brasilia, 2017). He is an adjunct professor at Department of Computing Science (CIC) at UnB where he is a tenured member of faculty. His research interests are network, information and cyber security.

**Matt Bishop** received his Ph.D. in computer science from Purdue University, where he specialized in computer security, in 1984. His main research area is the analysis of vulnerabilities in computer systems. The second edition of his textbook, Computer Security: Art and Science, was published in 2002 by Addison-Wesley Professional. He is currently a co-director of the Computer Security Laboratory at the University of California Davis.

**Marcelo Antonio Marotta** is an adjunct professor at the University of Brasilia, Brasilia, DF, Brazil. He received his Ph.D. degree in Computer Science in 2019 from the Institute of Informatics (INF) of the Federal University of Rio Grande do Sul (UFRGS), Brazil. His research involves Heterogeneous Cloud Radio Access Networks, Internet of Things, Software Defined Radio, Cognitive Radio Networks, and Network Security.