Adequação Online de Rastros de Tráfego de Rede nos Clientes para Alimentar Sistemas de Detecção de Intrusão

Otávio O. Silva, Daniel M. Batista

[1]Departamento de Ciência da Computação – Instituto de Matemática e Estatística
Universidade de São Paulo (USP) – São Paulo, SP – Brasil

otavio.ols@usp.br, batista@ime.usp.br

Abstract. *The constant emergence of new applications and zero-day attacks make public datasets for training IDSs quickly obsolete, justifying that organizations (re-)generate their private datasets frequently. This paper presents a proposal to capture traffic on devices with low computational power and adapt it to the expected format for training IDSs without storing the raw traffic. Experiments have shown that the proposal reduces the execution time by up to 76% in generating datasets, despite the possibility of packet loss in the captured traffic depending on the bit rate. As an additional contribution, the software that implements the proposal is available as free and open-source software.*

Resumo. *O surgimento constante de novas aplicações e de ataques de dia zero fazem com que datasets públicos para treinamento de IDSs tornem-se obsoletos rapidamente, justificando que organizações (re-)gerem seus próprios datasets privados com frequência. Este artigo apresenta uma proposta para capturar o tráfego em dispositivos com baixo poder computacional e adequá-lo para o formato esperado para treinar IDSs sem armazenar o tráfego bruto. Experimentos mostraram que a proposta reduz o tempo de execução em até 76% na geração de datasets, apesar da possibilidade de perda de pacotes do tráfego capturado a depender do volume de fluxo. Como contribuição adicional, o software que implementa a proposta é disponibilizado como software livre.*

1. Introdução

Sistemas de Detecção de Intrusão (IDSs - *Intrusion Detection Systems*) desempenham um papel fundamental na segurança de redes de computadores. Quando instalados em equipamentos de interconexão, como no *gateway* padrão da rede, os IDSs monitoram o tráfego de entrada e saída da Internet e detectam padrões conhecidos ou comportamentos fora do normal para alertar, ou eventualmente mitigar, possíveis ciberataques que estejam ocorrendo em diversas camadas da Arquitetura Internet.

Para que um IDS possua uma boa eficácia espera-se que ele apresente baixas taxas de falsos positivos, que ocorre quando há um alerta mas não há uma intrusão, e baixas taxas de falsos negativos, que ocorre quando há uma intrusão mas não há um alerta. No caso de detectar comportamentos fora do normal, tanto a indústria quanto a academia têm investido bastante nos últimos anos na utilização de técnicas de Inteligência Artificial, principalmente aprendizado de máquina [Neupane et al. 2022] [İş 2024].

IDSs baseados em aprendizado de máquina precisam ser treinados para serem capazes de diferenciar um tráfego normal, chamado de benigno, de um tráfego de ataque,

chamado de malicioso. Logo, um treinamento mal feito pode afetar significativamente a qualidade do IDS e esse treinamento depende de bons conjuntos de dados, ou *datasets* [Khraisat et al. 2019].

Existem diversas opções de *datasets* disponíveis publicamente que permitem realizar o treinamento dos IDSs, como é o caso do CIC-IoT-2023 [Neto et al. 2023], que possui dados de tráfego benigno e de diferentes tipos de ataques no contexto de redes com dispositivos da Internet das Coisas (IoT). O uso de *datasets* tradicionais, como o CIC-IoT-2023, permite que um novo IDS possa ser comparado mais facilmente com outros IDSs existentes, se eles foram avaliados com os mesmos *datasets*. Entretanto, com o surgimento constante de novas aplicações que geram padrões de tráfego diferente do convencional e de ataques de dia zero, treinar um IDS com dados de tráfego capturados há muito tempo atrás pode torná-lo inútil quando colocado em produção. Mesmo que o IDS seja inicialmente treinado com *datasets* atuais, retreiná-lo frequentemente com o tráfego real da organização é importante para que ele evolua juntamente com as novas ameaças.

Gerar *datasets* exige a captura do tráfego, o armazenamento desses dados brutos em um arquivo PCAP intermediário e a posterior transformação desse arquivo em um arquivo CSV com as características mais importantes para o modelo de aprendizado sendo usado. Como o compartilhamento do tráfego bruto carrega informações de camadas superiores que podem representar riscos à privacidade [Elias et al. 2022], pode exigir muito espaço em disco, e ainda como o que realmente importa é o CSV final, este artigo busca responder a seguinte pergunta de pesquisa: **Qual o desempenho de um mecanismo que gere datasets para treinamento de IDS sem a necessidade de armazenar o PCAP intermediário?** Além disso, capturar o tráfego nos dispositivos de borda ao invés de capturar no *gateway* padrão permite facilmente escalar a obtenção dos dados, coletando valores dos dispositivos da rede individualmente. No contexto de um IDS descentralizado, como uma rede IoT com aparelhos de baixa capacidade de processamento, essa abordagem se mostra útil. Nesse sentido, esse trabalho avalia a geração dos *datasets* em um Raspberry Pi Model 3B, dispositivo de processamento limitado.

Experimentos avaliando o desempenho da proposta e de outro mecanismo da literatura mostraram que foi possível reduzir o tempo de execução do processo de geração em até 76%. Entretanto, em volumes mais elevados de fluxo, há perda de parte do tráfego capturado. Uma contribuição adicional deste artigo é a disponibilização de um software livre que implementa a proposta [1], permitindo a reprodução dos experimentos e a evolução da ferramenta pela comunidade científica.

O restante deste artigo está organizado da seguinte forma: A Seção 2 descreve a proposta. A Seção 3 detalha o projeto de experimentos. A Seção 4 apresenta a análise de desempenho da proposta, comparando-a com outra ferramenta da literatura. O artigo é finalizado com conclusões e sugestões para trabalhos futuros na Seção 5.

2. Geração Online de CSV sem PCAP Intermediário

O método convencional para gerar *datasets* para treinamento de IDSs é simular a troca de pacotes entre os dispositivos e utilizar algum mecanismo ou dispositivo externo para monitorar os fluxos, usualmente registrando os dados em arquivos de formato PCAP (*Pac-*

[1] `https://github.com/otavioolsilva/wtg-2025`

ket Capture) ou PCAPNG (*PCAP Next Generation*)[2]. Todavia, apesar desse formato ser amplamente utilizado para estudos e análises, é uma prática comum sua conversão para outro mais acessível aos modelos conhecidos de aprendizado de máquina, como CSV (*Comma-Separated Values*). A mudança entre esses formatos não é realizada de maneira direta e envolve decisões de quais dados manter e como representá-los, pois estamos transformando um formato estruturado e adequado para pacotes de rede, os quais podem apresentar padrões dos mais diferentes, em uma tabela com campos fixos. Existem ferramentas já conhecidas capazes de extrair os campos úteis do PCAP e realizar a conversão, como o CICFlowMeter e ALFlowLyzer [Shafi et al. 2024]. Alguns *datasets* também são publicados junto com *scripts* próprios que foram utilizados para realizar essa conversão, como é o caso do material suplementar que acompanha o CIC-IoT-2023.

Essa abordagem para a produção de dados, entretanto, apresenta um revés: a conversão de arquivos PCAP para CSV é uma etapa intermediária necessária para a obtenção dos resultados almejados e que só pode ser realizada após a conclusão da simulação dos fluxos de rede, gerando um gasto de espaço, com arquivos que não são necessariamente o produto final, e de tempo, visto que as etapas são sequenciais. Uma possível solução para esse cenário é não mais fazer uso desse passo auxiliar e partir da simulação do tráfego de rede diretamente para os arquivos CSV finais. Dessa maneira, é possível ainda descentralizar a coleta dos pacotes realizada por um monitor de rede e performar essa ação nos dispositivos da ponta da rede de testes, os próprios clientes. Nas próximas seções, vamos discutir a viabilidade desse procedimento, realizando a análise de experimentos conduzidos com um dispositivo de baixa capacidade para verificar o quão impactante é o processamento do tráfego e o quão efetivo é esse método quando comparado com o usual.

3. Projeto de Experimentos

Para comparar o desempenho e efetividade dos dois métodos (com a criação do PCAP intermediário e sem a criação do PCAP intermediário), escolhemos utilizar os *scripts* Python do material suplementar do conjunto de dados CIC-IoT-2023, para processar o conteúdo dos pacotes capturados do tráfego de rede e gerar os arquivos CSV finais. A diferença entre os cenários se dá em como os pacotes são entregues a esse processamento: para os experimentos tomados como referência, coletamos os dados de rede e geramos arquivos PCAP, que depois foram processados para gerar os CSV da mesma maneira realizada para a produção do *dataset* que acompanha os *scripts* utilizados; nos casos de experimentos sem etapas intermediárias, modificamos o conjunto de códigos já citado para receber os dados diretamente do tráfego à medida que ele vai sendo capturado (*online*) por um *sniffer*, gerando o arquivo CSV final a partir dos pacotes em memória, sem escrevê-los em disco. Cada linha do CSV obtido apresenta métricas de um conjunto de no máximo 10 pacotes processados através da biblioteca Python dpkt. Os detalhes das métricas e de como são geradas podem ser encontrados em [Neto et al. 2023].

Para os experimentos de referência, primeiro os dados da rede são coletados através da biblioteca Pyshark, um *wrapper* Python para o *software* Tshark, e armazenados em um arquivo PCAP. Após a captura, os arquivos são entregues aos *scripts* do CIC-IoT-2023 para serem processados e obtermos o CSV. As únicas modificações realizadas nesses códigos de processamento foram adequações à estrutura do arquivo principal

[2]Sem perda de generalidade, no restante deste artigo será feita referência apenas a arquivos PCAP.

para transformá-lo em uma função invocável e adições de metrificadores de uso de CPU e memória na função que faz a extração de características dos pacotes de rede. Os arquivos PCAP são divididos em arquivos com 10MB cada e processados em paralelo através de diferentes processos, no máximo três executando ao mesmo tempo devido à limitação do desempenho do nosso dispositivo cliente. Cada processo reporta o seu uso de CPU e tempo de execução quando é encerrado, permitindo-nos somar esse valor à métrica de uso final tomando-o proporcionalmente ao tempo total de execução do código.

Para os experimentos de nossa proposta, ou seja, nos cenários que geram apenas o CSV, duas *threads* concorrem no código Python: uma realiza a coleta de pacotes da rede enquanto a outra os processa para gerar os CSV. Os dados da rede são coletados através da biblioteca Pypcap do Python e entregues para a outra *thread* através de um *buffer* compartilhado. Duas abordagens foram avaliadas: na primeira, entregamos os pacotes em uma janela de 10 em 10 com o *buffer* possuindo 5.000 posições ao todo (suficiente para todo o fluxo produzido), permitindo um processamento gradual; a outra abordagem entrega todos os pacotes coletados juntos para o processamento, de maneira que este só inicia após o término da captura. A intenção é estudar o quão eficiente é segmentarmos os dados de entrada e processá-los em volumes menores quando comparado com a escolha convencional de analisar todos os pacotes de uma única vez, mas ainda sem gerar um PCAP intermediário. O processamento dos pacotes é feito por uma versão modificada dos *scripts* que estamos utilizando para a geração do CSV, adequados para o novo formato de entrada de dados e removendo funções fora do escopo deste trabalho. Como cada entrada no CSV final se refere a um teto de 10 pacotes, a janela de tamanho 10 da primeira abordagem permite que o conjunto seja processado de uma única vez por inteiro, apresentando bons resultados nos experimentos preliminares.

Para executar as baterias de teste, um *script* em Bash foi utilizado para orquestrar o procedimento. Cinco iterações dos passos abaixo são realizadas:

- Executamos o código *Python* em segundo plano e aguardamos 20 segundos[3] para garantir que o *sniffer* esteja pronto.
- A ferramenta iperf3 é utilizada para gerar um fluxo de dados UDP segundo uma taxa de bits definida pelo experimento em questão. O fluxo dura 10 segundos.
- Na sequência, aguardamos até o código *Python* terminar o seu processamento dos dados e então encerramos a execução, apagando os arquivos gerados.

Os dados são gerados a partir dos resultados das cinco iterações, em geral com as métricas finais sendo a média de cada uma das execuções. As execuções do iperf3 se deram com *bit rates* de 5Mbits/s, 10Mbits/s, 15Mbits/s, 20Mbits/s e 25Mbits/s, para nos permitir perceber com clareza a partir de qual momento os métodos começam a se distanciar em termos de desempenho e efetividade. Nesse sentido, dado que a proposta deste artigo é investigar a performance de dispositivos de baixo poder computacional lidando com diferentes volumes de tráfego, todos os fluxos gerados nestes experimentos são benignos, visto que o conteúdo dos pacotes não é levado em consideração para análise. Os valores escolhidos foram suficientes para demonstrar o padrão de degradação da eficiência das bibliotecas consideradas, de maneira que ataques que atuam com um alto volume de pacotes tendem a reproduzir esse mesmo comportamento nas devidas proporções.

[3]No caso dos experimentos com os *scripts* de referência, esse valor foi alterado para 25 segundos, visto que o código Python leva um tempo considerável para iniciar a captura com o Pyshark.

Para realizar a coleta dos dados para posterior análise, várias ferramentas foram utilizadas. Para obter o consumo médio de CPU e a taxa média de leitura e escrita em disco, a biblioteca Python psutil foi utilizada (no caso da CPU, o consumo pode ser apresentado acima de 100% se houverem múltiplos processos sendo executados simultaneamente). Para mensurar o pico de uso de memória, o módulo Python resource foi utilizado. O tempo total de execução foi obtido através do utilitário bash time e o número de linhas do arquivo CSV final foi obtido através do comando wc.

Todos os experimentos foram conduzidos em uma rede local e sem acesso à Internet. Duas máquinas, conectadas diretamente através de um cabo Ethernet CAT 5e, foram utilizadas, tendo sido todos os dados estudados coletados no cliente. O cliente era um Raspberry Pi Model 3 B equipado com Quad Core 1.2GHz Broadcom BCM2837 64-bit CPU, 1GB RAM rodando Debian GNU/Linux 12 (bookworm) e com uma interface de rede de 100 Mbits/s. O servidor foi um Acer Aspire 3 A315-41-R4RB equipado com AMD Ryzen 5 2500U 2.0GHz 64-bit CPU, 12GB DDR4 2667 MHz RAM rodando Fedora Linux 41 (Silverblue) e com uma interface de rede de 1000 Mbits/s.

4. Resultados

Pela natureza do método que faz uso de arquivos PCAP intermediários, que é o método *baseline* que visamos aprimorar, nunca há perda dos pacotes que foram gerados pela simulação de tráfego, visto que o *sniffer* está em atuação constante durante todo o fluxo e registrando-o sem concorrer com qualquer outra atividade. Com a nossa proposta, sem o PCAP intermediário, temos que lidar com mudanças de contexto para entregar os pacotes para a *thread* que faz o processamento destes para gerar o CSV, resultando em algumas perdas, como é possível ver na Figura 1 que plota o percentual de perda de pacotes da nossa proposta em relação ao método tradicional (quanto menor o valor, melhor).

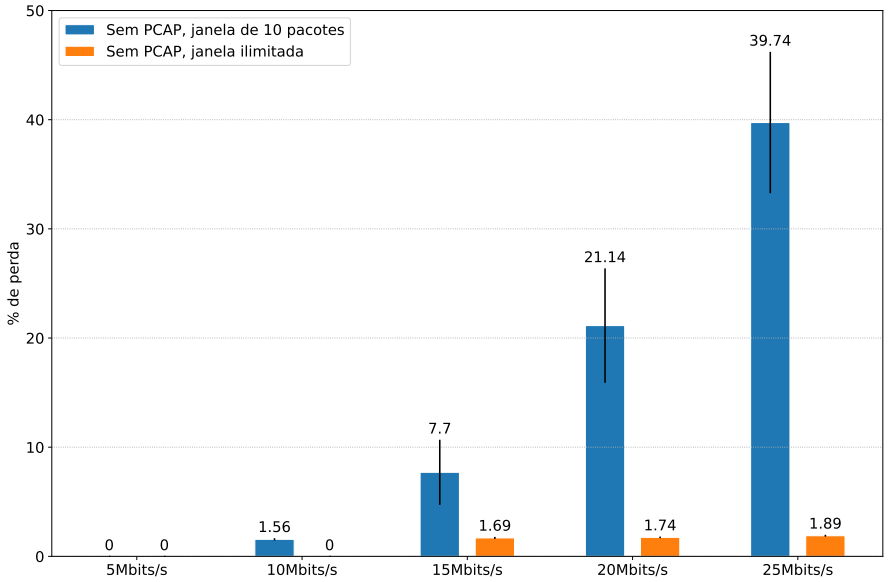

Figura 1. Percentual de perda média de pacotes. Obtido comparando o número de linhas dos arquivos CSV com o gerado pelos códigos de referência.

É notável que o aumento do fluxo de dados resulta em uma variação clara de efetividade do método sem PCAP intermediário e com janela de 10 pacotes. A degradação

do desempenho não é exatamente proporcional ao aumento do fluxo, mas correlato, e isto fica ainda mais evidente observando o desvio padrão, representado por uma linha preta no topo das barras: valores altos de taxa de transmissão implicam em um desvio alto, indicando grandes variações entre as diferentes execuções e inconstância do algoritmo. Por outro lado, quando ajustamos o nosso método para não utilizar a janela de 10 pacotes, o percentual de perda é baixo, de menos de 2% no cenário com transmissão a 25Mbits/s.

Essa diferença nos resultados obtidos vem com um alto custo computacional, conforme a Figura 2 indica. Por mais que o método que não faz uso de PCAP intermediário e com janela ilimitada seja tão efetivo quanto o método com PCAP intermediário em termos de perda de pacotes, o tempo de execução para o cenário de 25Mbits/s entre essas abordagens é quase o triplo em favor do que gera arquivos auxiliares. Há de se considerar que esse comportamento é esperado: o método com PCAP intermediário nativamente faz uso de paralelismo, utilizando até três processos para gerar os CSV a partir dos dados lidos do disco, valor proporcional à diferença que observamos aqui para o caso mais extremo, enquanto no outro as *threads* concorrem por um único núcleo do processador. Evidentemente, conforme pode ser observado na Figura 3, a diferença do uso de CPU entre esses métodos é elevado, sendo mais eficiente aquele que não faz uso do disco.

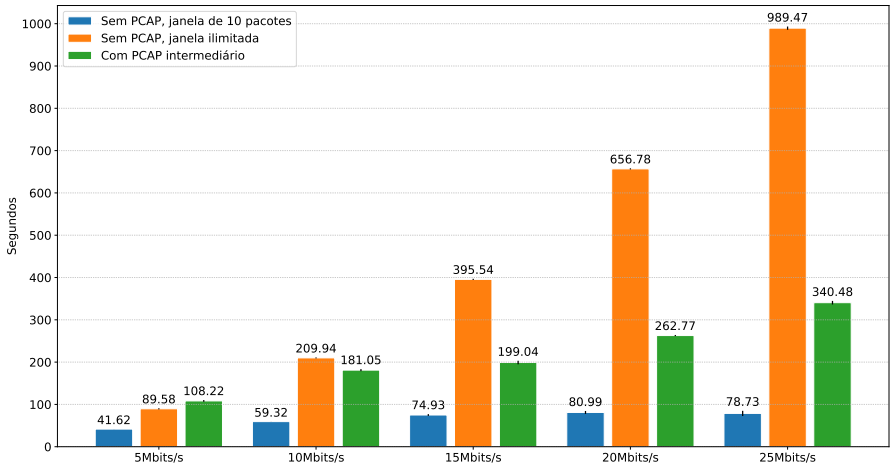

Figura 2. Tempo médio de execução.

O método sem PCAP intermediário e com janela de 10 pacotes é coerente no seu gasto computacional: à medida que o fluxo aumenta e a perda de pacotes se acentua, o seu consumo de CPU também atinge um platô, conforme observado na Figura 3. Observando os dados numéricos coletados, esse método foi capaz de processar até um valor de aproximadamente 1300 pacotes por execução, sendo que já em 15Mbits/s tivemos um total de 1300 pacotes trafegados. Assim, a baixa diferença de consumo de poder computacional entre os três casos mais extremos é justificável, visto que o número de pacotes processados entre os três também foi próximo. De maneira semelhante, podemos observar na Figura 2 que o tempo de execução entre esses três casos representado nas barras azuis é muito próximo. Entretanto, esse tempo consumido não é proporcional à perda: no caso de 15Mbits/s, temos uma perda de 7%, mas um tempo de execução mais de duas vezes melhor que o método com PCAP intermediário. Assim, apesar da baixa efetividade, o desempenho em consumo de CPU é proporcionalmente melhor.

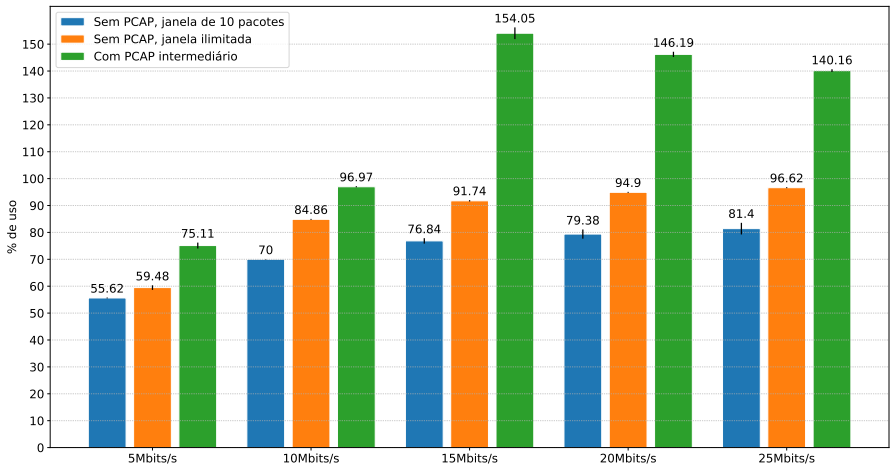

Figura 3. Uso médio de CPU. Valores acima de 100% indicam o uso de mais de um núcleo do processador do Raspberry simultaneamente.

Ainda sobre o uso de CPU, é notável que os valores obtidos no método com PCAP não se mostraram crescentes conforme o aumento da taxa de bits trafegando. Essa observação é natural, considerando que a métrica foi avaliada tomando-se o consumo de cada processo proporcional ao tempo total e, como o tempo de execução cresceu e não necessariamente o número de processos aumentou, o uso médio do processador decresceu. Mas note que a diferença entre os valores é próxima e ainda assim o tempo aumentou significativamente, de maneira que o esforço computacional foi maior entre os casos de taxa de bits mais elevadas. Esse esforço se evidencia ao observamos o uso de memória na Figura 4: o método tradicional foi o que apresentou maior consumo deste recurso, possivelmente limitado para os clientes. Este método também se destacou no uso de disco, entretanto os valores registrados apresentaram alto desvio padrão visto que foram computados a nível de sistema e não de processo, dificultando uma análise conclusiva. Estudar o impacto em disco das abordagens é uma proposta para trabalhos futuros.

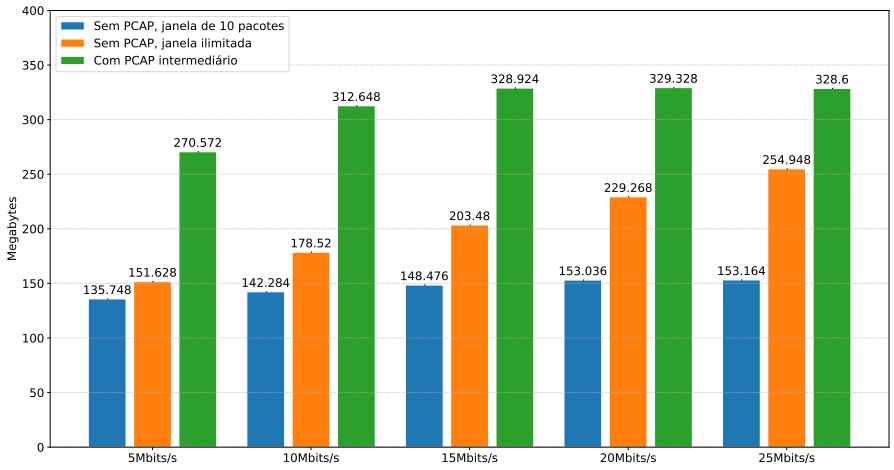

Figura 4. Uso máximo de memória entre todos os processos durante a execução.

Observando a Figura 4, manter todos os dados na memória do dispositivo e não gravá-los em disco não gerou um grande impacto no consumo de RAM disponível em

contraste com o método tradicional. Além disso, processar os pacotes em janelas de 10 em 10 surtiu um efeito considerável quando comparado com a janela ilimitada, com um aumento gradual bem menos expressivo à medida que a taxa de transmissão aumenta.

5. Conclusões e Trabalhos Futuros

Respondendo à pergunta de pesquisa apresentada na introdução: em termos de efetividade do processamento dos dados, o método tradicional, que faz uso de um arquivo PCAP intermediário, se mostrou o melhor no sentido de não haver perdas no arquivo CSV final. Próximo deste resultado, nossa proposta sem o PCAP e com janela ilimitada demonstrou uma baixa perda de pacotes. Limitar a janela de pacotes que são entregues para processamento levou a ganhos positivos: o tempo de execução é reduzido drasticamente e tanto os gastos de CPU quanto de memória se mostraram mais baixos. Em contrapartida, nota-se que pode haver perda significativa de pacotes no arquivo final. Assim, essa abordagem se mostra viável computacionalmente para baixas taxas de tráfego de bits em dispositivos IoT de borda, sendo a redução do número de pacotes perdidos um ponto interessante de pesquisa futura, entendendo o impacto do aumento do poder computacional ou paralelizando nossa proposta, dessa maneira acabando com o problema da concorrência entre a *thread* que faz a captura e a que realiza o processamento dos pacotes.

Agradecimentos

Esta pesquisa é parte do projeto de pesquisa STARLING – Segurança e Alocação de Recursos em B5G via Técnicas de Inteligência Artificial, um projeto selecionado na chamada pública FAPESP/MCTI/MCom/CGI.br 2022 (proc. 2021/06995-0). Ela também é parte do projeto FAPESP proc. 2024/10240-3.

Referências

Elias, E. M. d., Carriel, V. S., De Oliveira, G. W., Dos Santos, A. L., Nogueira, M., Junior, R. H., and Batista, D. M. (2022). A Hybrid CNN-LSTM Model for IIoT Edge Privacy-Aware Intrusion Detection. In *Anais do IEEE LATINCOM*, pages 1–6.

Khraisat, A., Gondal, I., Vamplew, P., and Kamruzzaman, J. (2019). Survey of Intrusion Detection Systems: Techniques, Datasets and Challenges. *Cybersecurity*, 2(20).

Neto, E. C. P., Dadkhah, S., Ferreira, R., Zohourian, A., Lu, R., and Ghorbani, A. A. (2023). CICIoT2023: A Real-Time Dataset and Benchmark for Large-Scale Attacks in IoT Environment. *Sensors*, 23(13).

Neupane, S., Ables, J., Anderson, W., Mittal, S., Rahimi, S., Banicescu, I., and Seale, M. (2022). Explainable Intrusion Detection Systems (X-IDS): A Survey of Current Methods, Challenges, and Opportunities. *IEEE Access*, 10:112392–112415.

Shafi, M., Lashkari, A. H., and Mohanty, H. (2024). Unveiling Malicious DNS Behavior Profiling and Generating Benchmark Dataset through Application Layer Traffic Analysis. *Computers and Electrical Engineering*, 118:109436.

İş, H. (2024). A Comprehensive Analysis of NGFWs for Cyber-Physical System Security After the CrowdStrike Incident. In *Anais da Global Energy Conference (GEC)*, pages 12–20.