Detecção de Ataques DDoS em Tempo de Execução baseado em Modelo Transformer Otimizado por Chunks

Gustavo F. Pereira[1], **Euclides Peres Farias Jr.**[2], **Anderson Berganini de Neira**[1,3], **Michele Nogueira**[1,2]

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)

[2]Departamento de Informática – Universidade Federal do Paraná (UFPR)

[3]Instituto Federal do Paraná (IFPR) - Quedas do Iguaçu, Brasil

gfp@ufmg.br, epfjunior@inf.ufpr.br,

anderson.neira@ifpr.edu.br, michele@dcc.ufmg.br

Abstract. *Distributed Denial of Service (DDoS) attacks are becoming increasingly faster to execute and more harmful. Thus, detecting them quickly becomes crucial. Solutions that employ robust AI models have inference times of several seconds or minutes in attack detection. This article proposes a DDoS attack detection method based on chunks with a Transformer architecture, enabling detection over network streams at runtime. The method captures traffic and processes it in time windows decomposed into smaller windows, reducing computational cost and classifying the window in a binary manner. In the experiments, the method achieved 99% accuracy, and the identification of malicious windows was performed in 35 ms.*

Resumo. *Os ataques de negação de serviço distribuídos (do inglês, Distributed Denial of Service – DDoS) estão cada vez mais rápidos em sua execução e mais prejudiciais. Assim, detectá-los rapidamente se torna crucial. As soluções que empregam modelos robustos de IA possuem tempos de inferência de vários segundos ou minutos na detecção de ataques. Este artigo propõe um método de detecção de ataques DDoS baseado em chunks com uma arquitetura Transformer, permitindo a detecção em fluxo de rede em tempo de execução. O método captura o tráfego e o processa em janelas temporais decompostas em janelas menores, reduzindo o custo computacional e classificando a janela binariamente. Nos experimentos, o método alcançou 99% de acurácia, e a identificação de janelas maliciosas foi realizada em 35 ms.*

1. Introdução

No mundo hiperconectado, os ataques de Negação de Serviço Distribuído (do inglês, *Distributed Denial of Service* - DDoS), estão cada vez mais presentes na Internet, afetando diversos setores da indústria, serviços bancários e provedores de Internet. Em uma de suas variações conhecida como *flood*, esses ataques inundam um alvo como servidores e provedores, com tráfego malicioso baseado em diferentes protocolos de rede, impedindo que o servidor responda a requisições legítimas [Holdsworth and Kosinski nd]. Consequentemente, o serviço torna-se indisponível. Diversos incidentes, tanto recentes quanto antigos, evidenciam o impacto desse tipo de ataque. Em outubro de 2025,

por exemplo, ocorreu um ataque de inundação baseado em tráfego UDP direcionado a um servidor de jogos online. O ataque atingiu um volume de aproximadamente 6 Tbps durante 30 a 40 segundos [Gcore 2025]. Um grupo intitulado Killnet pró-Ucrânia cometeu diversos ataques DDoS com base em requisições do protocolo HTTP contra vários serviços públicos da Lituânia em 2022, deixando mais de 100 serviços fora do ar. Em seus relatórios oficiais, o governo da Lituânia temia uma escalada de conflito para um conflito armado por causa desse ataque, mostrando o quanto esses ataques são danosos não somente por tirar serviços da rede, mas também por causar maiores consequências [of the Republic of Lithuania 2022].

Há diversos trabalhos na literatura dedicados à detecção de ataques DDoS, utilizando modelos de Inteligência Artificial (IA) ou abordagens matemáticas. O estudo de [Li et al. 2025] emprega o modelo Transformer aliado a características dos pacotes de rede e dependências temporais para melhorar as métricas de detecção em comparação a outros métodos, embora não tenha como foco a detecção em tempo hábil para posterior mitigação do ataque. O artigo [Ashraf et al. 2025] investiga diferentes modelos de *machine learning – ML*, variando desde algoritmos mais simples, como *Decision Tree*, *Logistic Regression* e *Naïve Bayes*, até modelos mais robustos, como *Support Vector Machine* e *Artificial Neural Networks*. Os autores mostram que modelos mais leves alcançam baixas latências de detecção, porém apresentam limitações de generalização. Por outro lado, os modelos mais robustos tendem a generalizar melhor em cenários variados, mas possuem tempos de inferência significativamente maiores. Outra estratégia para detectar múltiplos tipos de ataques, não apenas DDoS, é apresentada em [Ullah et al. 2024], que utiliza um modelo Transformer do tipo *Bidirectional Encoder Representations from Transformer – BERT* Large combinado com técnicas adicionais, como *Convolutional Neural Network – CNN*, *Recurrent Neural Networ – RNN* e o método de balanceamento SMOTE, a fim de avaliar quais combinações produzem melhores resultados. No entanto, o uso de modelos de grande porte acarreta custos computacionais elevados, impactando diretamente na velocidade de detecção e na viabilidade de aplicações em tempo de execução.

As abordagens de detecção de ataques DDoS têm incorporado, de forma crescente, técnicas de IA. Entretanto, muitos dos modelos apresentam alta complexidade computacional e, apesar de alcançarem desempenho expressivo, não foram projetados para operar em tempo de execução [Mittal et al. 2023]. Considerando que ataques DDoS costumam esgotar os recursos da vítima em intervalos extremamente curtos, torna-se necessário empregar técnicas capazes de generalizar o comportamento dos ataques para diferentes cenários e, ao mesmo tempo, oferecer respostas rápidas o suficiente para permitir ações de mitigação antes que o serviço seja comprometido [Dantas Silva et al. 2020]. Para resolver essas questões, este artigo propõe um método de detecção de ataques DDoS embasado em um modelo *Transformer* otimizado, composto por apenas um *encoders* com duas camadas e oito cabeças de atenção. O método processa o tráfego de rede dividido em *chunks*, ou seja, janelas menores com 120 endereços cada. Posteriormente, esses *chunks* são recombinados para formar uma janela completa. Essa estratégia permite analisar o comportamento da rede a partir de pequenas amostras e, paralelamente, decidir em tempo de execução se alguma dessas *chunks* apresenta indícios de ataque.

Avalia-se o método a partir de quatro experimentos realizados. Cada um dos experimentos aborda ataques DDoS com dois protocolos diferentes, sendo eles TCP Flood,

UDP Flood, e outro experimento com dois protocolos diferentes, TCP e UDP. Utilizando janelas temporais de 5 segundos e 2 minutos de tráfego de rede dividido em normal e malicioso, os resultados foram, em média, de 150 ms para análises de janelas temporais completas e, para janelas temporais que continham ataque, média de 35 ms. Para métricas como acurácia e precisão, com os protocolos UDP, TCP e TCP e UDP juntos, atingiu-se 99% de acurácia e precisão.

Este artigo procede como segue. A Seção 2 apresenta os trabalhos relacionados, artigos com o mesmo objetivo ou com proposta semelhantes. A Seção 3 descreve o método proposto descrevendo treino, pré-processamento, *features* usadas, teste e análise de tráfego de rede. A Seção 4 discute a avaliação e os experimentos realizados, levando em consideração os processos de avaliação e métricas. Por fim, a Seção 5 reúne as conclusões e considerações finais deste trabalho.

2. Trabalhos Relacionados

Existem trabalhos na literatura relacionados à detecção de ataques DDoS e DoS. Alguns abordam tanto versões estáticas com equações matemáticas e lógica difusa, quanto o uso de IA para a tarefa de detecção de ataques. No artigo [Ashraf et al. 2025], o autor seleciona as melhores *features* no *dataset* Bot-IoT para realizar os testes, onde são escolhidos modelos de IA, tendo como modelos com tempos mais expressivos o *Naïve Bayes*, o *Decision Tree* e o *Logistic Regression*, que apresentam valores de até 1 a 5 segundos para processar 1 milhão de endereços. Esses testes não levam em consideração os tempos de pré-processamento do *dataset*, tratando-os como uma fase à parte. Porém, esses modelos apesar de terem complexidade baixa, eles são limitados em relação à generalização, tendo desempenhos piores com outros protocolos e diferentes tipos de ataques.

Os modelos *Transformer* representam uma das arquiteturas mais influentes em aprendizado profundo, baseando-se no mecanismo de *self-attention* para capturar relações de longo alcance em sequências. A forma original, proposta em [Vaswani et al. 2017], utiliza uma estrutura *encoder-decoder*, na qual o *encoder* aprende representações compactas da entrada enquanto o *decoder* gera a saída condicionada a essas representações, sendo amplamente empregada em tarefas de tradução. O modelo é altamente paralelizável além de comportar várias alterações em sua arquitetura.

O trabalho [He et al. 2024] apresenta uma logica de combinação de captura de pacote e análise da rede por janelamento para a detecção de ataques DDoS. O modelo proposto para a detecção se baseia em uma versão reduzida de uma CNN para conseguir distinguir o tráfego benigno e malicioso com o auxilio tanto das informações dos pacotes, quanto as informações das janelas temporais, alcançando resultados de 99% em acurácias nos *dataset* da literatura . Porém o modelo usado tem acesso a poucos pacotes que constituem a janela, sendo de 3 a 11, o que pode gerar percas de contexto entre os pacotes, podendo levar a detecções errôneas em ataques que demandão maior contexto no tráfego.

O artigo [Najar and Manohar Naik 2025] propõe o uso de um modelo híbrido para realizar a detecção dos ataques combinando extração de informações de pacotes e modelagem temporal. A estrategia abordada no artigo foi combinar um CNN para a análise de pacotes e uma *Bidirectional long-short term memory – Bi-LSTM* junto de uma camada de atenção para analisar as características temporais da rede para conseguir fazer detecções de forma assertiva com resultados de 99% de acurácia em media e alguns ataques che-

gando a 100%. Entretanto um modelo que possui duas redes neurais *Long Short Term Memory – LSTM*, aumentando muito o custo computacional empregado nessa solução.

No campo da detecção de ataques DDoS, o estudo abrangente apresentado em [Farias et al. 2025] destaca o potencial dos modelos baseados em *Transformer*, sobretudo pela capacidade de capturar dependências complexas entre fluxos e de modelar simultaneamente padrões espaciais e temporais do tráfego. Segundo os autores, o mecanismo de atenção é capaz de identificar relações sutis entre pacotes e fluxos que caracterizam anomalias e assim detectam ataques, mesmo quando o tráfego aparenta normalidade em janelas locais. O trabalho ressalta que, quando otimizados, os modelos *Transformer* operam em tempo real, aproveitando a paralelização intrínseca ao *self-attention* para reduzir a latência e aumentar a eficiência na detecção de ataques DDoS.

No artigo de [Hernandez et al. 2025], são feitos vários testes com 3 modelos diferentes, sendo CNN, LSTM, e *Transformer*, para medir as métricas clássicas como F1-score, acurácia, *recall*, precisão e também os tempos de inferência dos modelos em janelas de 1 segundo. Foram realizados vários experimentos usando o *dataset* conhecido pela literatura, mas adicionando arquivos do MAWI para aumentar a semelhança com tráfegos reais. Apresenta resultados do Transformer variando de 0,7 segundos a 2,5 segundos, os autores então comentam que o modelo Transformer usado não tem tempo adequado para detecção, apesar de boas métricas. Porém, o modelo tem muitos parâmetros como quantidades de *encoders* e dimensão. Um modelo reduzido pode melhorar o tempo, assim como o processamento em *chunks*.

No artigo de [Wang and Li 2021], é proposto um modelo híbrido de *Transformer* para operar no plano de controle de uma rede *Software Defined Network – SDN*. São realizados testes com esse e outros modelos usados em trabalhos com SDN, como *Recurrent Neural Network – RNN*, *Gated Recurrent Unit*, LSTM, modelos híbridos e *Bidirectional Gated Recurrent*. Todos têm suas métricas avaliadas com o *dataset* CICDDoS2019. O modelo híbrido de *Transformer* com CNN foi o que apresentou melhores resultados de acurácia e F1-score, de 99%, atestando a eficácia do modelo. Porém, nenhuma métrica de tempo é mencionada no artigo, não sendo possível saber o tempo de processamento do modelo. O trabalho apresenta alto custo computacional, aliando o modelo *Transformer*, que por natureza é um modelo robusto, com outro, aumentando significativamente suas complexidades.

No artigo [Nalayini et al. 2025], é proposto um sistema de detecção de intrusões para redes SDNs que combina, em uma única arquitetura, a seleção de atributos quântico-inspirada, um modelo *Transformer* espaço-temporal, aprendizado por reforço hierárquico e aprendizado federado entre controladores. Os treinos são feitos usando o *dataset* CICIDS 2019 e depois é construída uma rede SDN para reproduzir esse *dataset* e então fazer as detecções, tendo resultados de 99% de acurácia, F1-score, *recall* e precisão, com uma média de 2,1 segundos de detecção a cada 1000 pacotes. Porém esse modelo junta outros sofwares e métodos de ML que seriam removidos para melhorar o tempo de detecção, além também da redução do tamanho do Transformer somente, que apresenta 6 encoder com 8 camadas de atenção cada, sendo o método mais custoso do artigo.

O artigo [Ali et al. 2025] usa um *Vision Transformer* composto exclusivamente por módulos de *encoder*, operando com hiperparâmetros ajustados para imagens de

$46 \times 46 \times 3$ pixels e classificando 23 variantes de ataques DoS/DDoS ao converter fluxos de rede em representações espaciais. A arquitetura segue o pipeline típico de *Vision Transformer*, envolvendo a divisão da imagem em *patches*, projeção linear por meio do *patch embedding*, adição do *positional encoding* e processamento por múltiplas camadas de autoatenção. O modelo apresenta bons resultados de acurácia e precisão, além de melhorias no tempo de inferência em relação a arquiteturas mais pesadas, contudo, o processamento de imagens tende a ser mais custoso do que o uso direto de tensores contendo informações dos pacotes, o que pode impactar a viabilidade em cenários de detecção em tempo de execução.

3. Método

Esta seção descreve o método proposto para a detecção de ataques DDoS em tempo de execução. A Figura 1 apresenta o método projetado em três etapas: (i) a captura do tráfego, (ii) o janelamento e (iii) a resposta do modelo. A primeira etapa consiste na captura do tráfego de rede para a criação dos arquivos de treino e análise de tráfego de rede. A próxima etapa descreve como é realizado o janelamento, a forma como os *chunks* são criados. A terceira etapa descreve como é obtida a elaboração e a resposta do modelo *Transformer*. As próximas subseções tratam das três etapas destacadas.

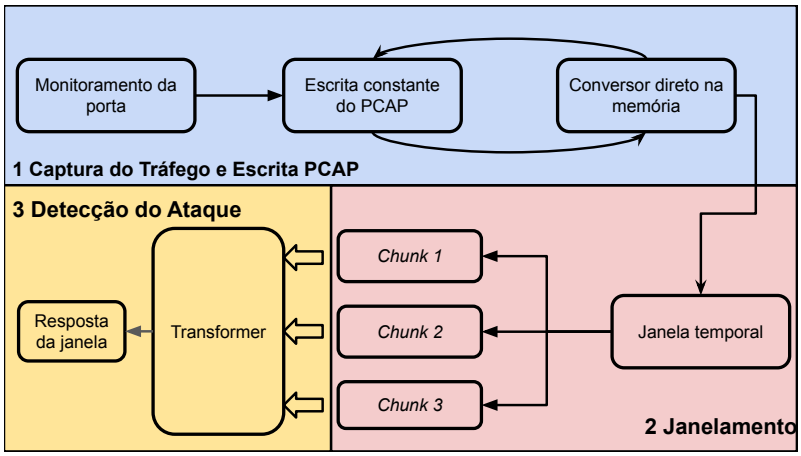

Figura 1. Processo completo de análise de tráfego

3.1. Captura do tráfego

A captura do tráfego da rede visa coletar informações dos cabeçalhos dos pacotes. Todo o tráfego da rede do usuário deverá ser coletado pelo método tem de ser replicado para uma porta específica *switch* com essa tecnologia. Deste modo, o método proposto não precisa mudar a topologia de rede. O método coleta os dados a serem analisados, que foram replicados na porta do *switch*. Após receber os dados, o método utiliza programas de captura de tráfego como *Tshark*, *Wireshark* ou *tcpdump* para coletar os dados. O método grava os dados capturados em um arquivo PCAP (do inglês, *Packet Capture*). O método utilizará o arquivo PCAP tanto para treinamento (Etapa 2), quanto para a detecção dos ataques (Etapa 3).

Após a coleta do tráfego de rede, o método converte o arquivo PCAP para um arquivo separado por vírgulas (do inglês, *Comma-Separated Values - CSV)*. Esta ação é

realizada apenas para proporcionar o treinamento do modelo (Subseção 3.3.1). Para a análise de tráfego de rede (Seção 3.3.2), a conversão é semelhante ao treinamento. Contudo, ao invés da conversão para CSV, o método proposto armazena os pacotes de rede em uma variável na memória do computador até atingir uma janela temporal (detalhada na próxima subseção). Quando esse processo se repete a janela temporal anterior é descartada e uma nova é armazenada. Esse processo é utilizado para diminuir o tempo presente na escrita e leitura de arquivos CSV e, como este trabalho considera a avaliação do tempo total de processamento, desde a transformação da janela em chunks, o pré-processamento dessas chunks e, por fim, a inferência do *Transformer*, a etapa adicional de escrita e leitura de arquivos CSV ampliaria o tempo medido quando comparado ao armazenamento e acesso em memória.

Após a conversão do PCAP, o método extrai as características que o modelo utiliza para aprender o funcionamento dos ataques e, consequentemente, detectar o ataque DDoS (Subseção 3.3). As características usadas no treinamento variam conforme o objetivo desejado. Elas podem ser focadas no tráfego de rede ou nas características individuais dos pacotes. Para a detecção de ataques DDoS (Subseção 3.3), as características coletadas são as que o cabeçalho do pacote permite extrair, sendo diferentes para cada protocolo abordado, como tamanho de pacotes ou características genéricas, como: média de pacotes coletada em um espaço de tempo ou média do tamanho dos pacotes em uma determinada janela de tempo. As características que forem usadas na análise de tráfego de rede tem que ser as mesmas usadas no treinamento do modelo.

3.2. Janelamento

O janelamento do tráfego é usado para que o tráfego seja dividido em *chunks* que constituem uma quantidade de endereços de rede que, juntos, constituem uma janela de tempo. Para que possa ser aproveitado as características gerais da rede em um intervalo de tempo. A estratégia de *chunks* diminui o tempo de análise de uma janela temporal. Esse tempo diminui a medida em que uma janela temporal é quebrada em vários *chunks* e esses *chunks* são analisados paralelamente, desse modo o modelo analisa pequenos blocos da janela até completar a janela, ao invés de analisar a janela toda de uma vez. A Figura 2 Demonstra como é o processo de transformação da janela em lotes e, posteriormente, a análise das *chunks* em um lote e a detecção de tráfego normal ou de ataque nessas *chunks*. As *chunks* são criadas imediatamente após a *chunks* anterior, evitando perda de contexto ou pacotes entre elas. Caso não haja endereços suficientes para completar uma *chunk*, ela é completada até o tamanho selecionado com o último endereço conhecido. Primeiro, os pacotes de uma janela temporal são armazenados em uma variável na memória. Isso é feito para descartar o processo de escrita e leitura de arquivos e assim poupar tempo. Cada *chunk* tem uma quantidade de endereços presente nela e cada janela temporal tem outra quantidade de *chunks* presentes nela.

O número de *chunks* e seus tamanhos variam de acordo com alguns fatores. O número de pacotes em um *chunk* varia conforme a capacidade computacional empregada: quanto mais endereços forem inseridos em uma *chunks*, mais o modelo irá demorar para inferir o resultado nessa *chunks* separadamente, porém, haverá maior conexão e compartilhamento de informação entre os dados capturados. É preciso encontrar um número de endereços que acelere o processo de inferência, mas que, ao mesmo tempo, não reduza a taxa de acerto ao utilizar poucos endereços por *chunks*.

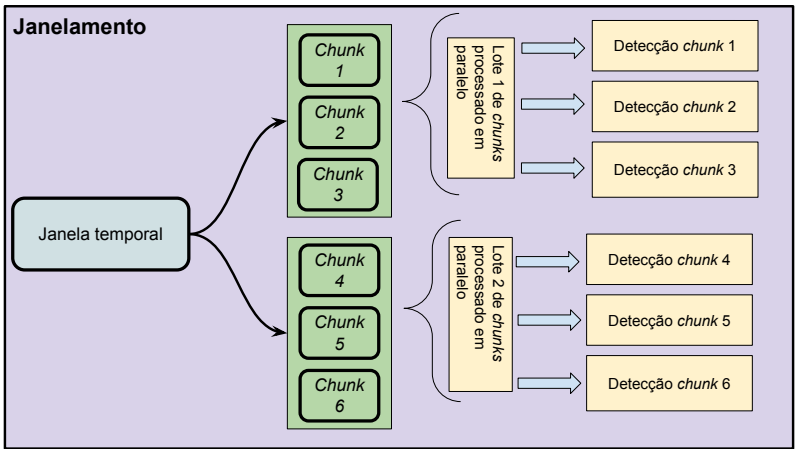

Figura 2. Processo da criação das *chunks*

Para a quantidade de *chunks* processados em paralelo, adota-se o mesmo princípio do aproveitamento do poder computacional do parágrafo anterior. O diferencial é que não há perda de contexto em relação ao número de *chunks* processadas, pois cada uma delas é avaliada por si própria, em paralelo. No entanto, o gargalo de transferência de dados da CPU para a GPU e não há necessidade de processar todas as *chunks* para obter a classificação do modelo. Por esse motivo, o processamento integral de todas as *chunks* não é a forma mais eficiente. Um exemplo disso é que o ataque pode estar no começo da janela temporal, de modo que as primeiras *chunks* já apresentariam indicações do ataque. Processar a janela completamente faz com que a inferência demore mais do que processar lotes de *chunks*, chegando ao limite de tempo que é possível reduzir, pois ainda há o gargalo de transferência de dados da CPU para a GPU. Em casos que o ataque esta no fim da janela o modelo irá processar lotes de *chunks* até chegar em uma *chunk* com ataque, ou o começo de outra janela temporal caso não encontre. A Figura 3 demonstra a identificação de um ataque que acontece na última *chunk* de uma janela temporal, após ter passado por toda a janela com chunks benignas.

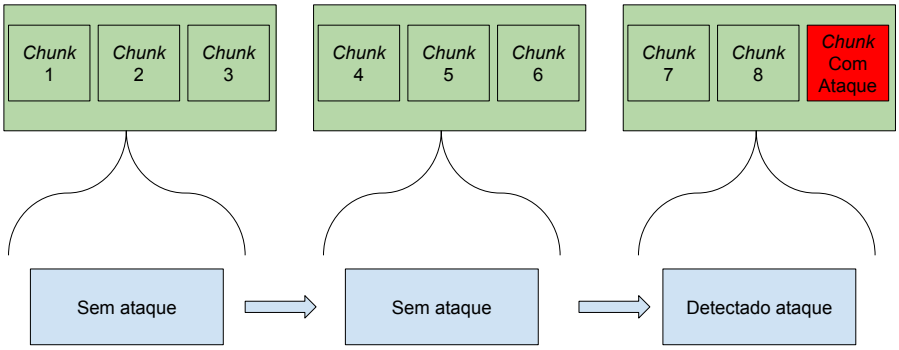

Figura 3. Detecção com *chunk*infectada no final da janela

3.3. Detecção de ataques DDoS em tempo de execução

Para realizar a detecção de ataques DDoS em tempo de execução, este trabalho utiliza a arquitetura dos modelos *Transformer*. O modelo *Transformer* construído neste traba-

lho possui uma arquitetura reduzida em relação ao *Transformer* original, com foco na otimização dos processos computacionais. Ele é constituído de apenas um bloco *encoder* com duas camadas, descartando o uso de *decoders* pela ausência da necessidade de recriação da janela. A primeira camada do *encoder* se baseia em compreender as características de cada *chunk*. A segunda camada finaliza a análise e decide se ela contém ataque ou não. Essa lógica existe para evitar excesso de falsos positivos, pois a regra de classificação adotada é que qualquer endereço de ataque presente em uma *chunk* transforma a *chunk* inteira em ataque, mesmo que ela seja majoritariamente composta por endereços benignos.

Cada pacote na janela é representado por um vetor de dimensão das *features* de cada pacote, que é projetado para um espaço de 64 dimensões por uma camada linear inicial. Em seguida, a sequência projetada é processada pelo módulo *encoder* com quatro cabeças de atenção, *layer dropout* de 0,1 representado pela Figura 4. Esses valores foram escolhidos por terem melhores resultados com tempo sem prejudicar as métricas de acerto. São processados ao mesmo tempo uma quantidade de pacotes que constituem uma *chunk*. Os *logits* gerados pelo modelo passam por uma função *sigmoid*, que produz à probabilidade da janela conter um ataque DDoS. Caso essas probabilidades estejam maiores que 90% de certeza essa *chunk* é considerada como um ataque.

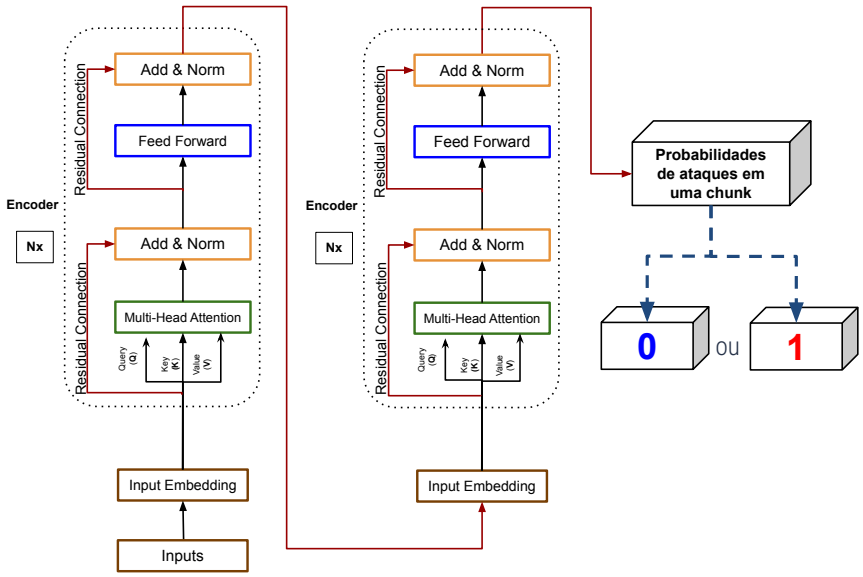

Figura 4. Arquitetura Usada. Adaptado de [Farias et al. 2025]

3.3.1. Treinamento do modelo transformer otimizado

Com os dados coletados é preciso fazer um pré-processamento para evitar dados que possam prejudicar o modelo. O pré-processamento consiste em substituir valores infinitos, positivos ou negativos e valores faltantes pelas médias das respectivas colunas. Nos casos em que vários protocolos são agregados no mesmo arquivo, atributos inexistentes para

determinado protocolo são preenchidos com zero. Para rotular os pacotes de ataque ou benignos, utiliza-se do conhecimento prévio dos dispositivos benignos e maliciosos, rotulando todo tráfego proveniente dos endereços benignos como normal, e dos endereços maliciosos como ataques. A rotulagem ajuda com que o modelo tenha mais precisão nas detecções dos ataques assim como um controle maior dos dados passados e aprendidos. Por fim, os dados são convertidos para tensor.

Após a etapa de treino é realizado um teste. Todo esse procedimento ainda é realizado de forma estática, sem medições de tempo nesta fase, o objetivo é verificar de maneira geral a capacidade do modelo, garantindo que ele não produza falsos positivos em excesso que comprometam a análise. Com esse teste, assegura-se que o modelo é capaz de analisar janelas temporais completas sem rotular *chunks* verdadeiramente benigna como ataque, preservando a confiabilidade da avaliação global. Uma vez obtidos resultados satisfatórios, o modelo é então encaminhado para a fase de análise de tráfego de rede em tempo de execução, caso contrário é preciso treinar novamente o modelo.

3.3.2. Análise de tráfego de rede

Para a análise de tráfego de rede, adotou-se uma abordagem que se baseia na ferramenta *tshark*, que permite extrair diretamente os atributos requeridos para a construção das janelas de análise [Wireshark 2026]. Nos casos em que determinados atributos não puderam ser obtidos diretamente, são realizados cálculos complementares para o povoamento dos campos faltantes. Esses dados são então convertidos em tensores com as mesmas dimensões e organização utilizadas na etapa de treinamento, garantindo compatibilidade com o modelo. Essa estratégia possibilita que todo o fluxo de dados seja mantido em memória, eliminando a necessidade de operações de escrita em disco.

Caso qualquer *chunk* apresente indícios de atividade maliciosa, a inferência da janela é imediatamente interrompida, e o sistema passa para a análise da próxima janela temporal. Por outro lado, se nenhuma anomalia é detectada durante a avaliação parcial, a janela inteira é examinada até o final. O modelo sempre dá as resposta depois de toda a janela ser processada, seja ela uma janela completa (sem ataque) ou uma janela que teve sua execução interrompida antes pela existência de ataque. A resposta do modelo é a classe escolhida para representar cada um dos cenários, como, por exemplo: 0 para tráfego normal e 1 para tráfego de ataque. Assim, o administrador de rede decide a atitude a ser tomada para lidar com o ataque.

4. Avaliação de desempenho

Esta seção apresenta a descrição e os resultados dos experimentos realizados. Inicialmente, descreve-se como os experimentos deste trabalho foram conduzidos, abrangendo a rede SDN utilizada e os processos de treino, teste e resposta da janela. Esta avaliação utilizou um Dell Vostro 7620, com 16 GB de memória RAM, uma placa de vídeo RTX 3050 Ti com 4 GB de memória e um processador Intel Core i7 de 12ª geração. Todos os resultados e modelos estão disponíveis online[1].

[1] `https://github.com/pacoca015/SBRC_2026`

4.1. Características gerais dos experimentos

Essa subseção representa as características usadas em todos os experimentos. A Figura 5 representa a topologia de rede SDN, emulada através da Mininet, empregada neste trabalho, que utiliza o controlador RYU para gerenciar o fluxo de tráfego na rede. A infraestrutura é composta por cinco *switches*, cada um associado a uma sub-rede contendo dez dispositivos, totalizando 50 dispositivos finais conectados que são alterados a cada experimento, tanto para treino, teste e monitoramento. Entre esses dispositivos, um é dedicado à execução de um servidor *iperf*, responsável por receber tanto o tráfego benigno quanto os diferentes vetores de ataque gerados nos experimentos. Para a simulação de cenários adversos para cada um dos ataques, seleciona-se um subconjunto de dispositivos distintos para tráfego benigno e malignos destinado para o servidor. Essa configuração possibilita avaliar o modelo em um ambiente controlado, que simule uma rede real *online*.

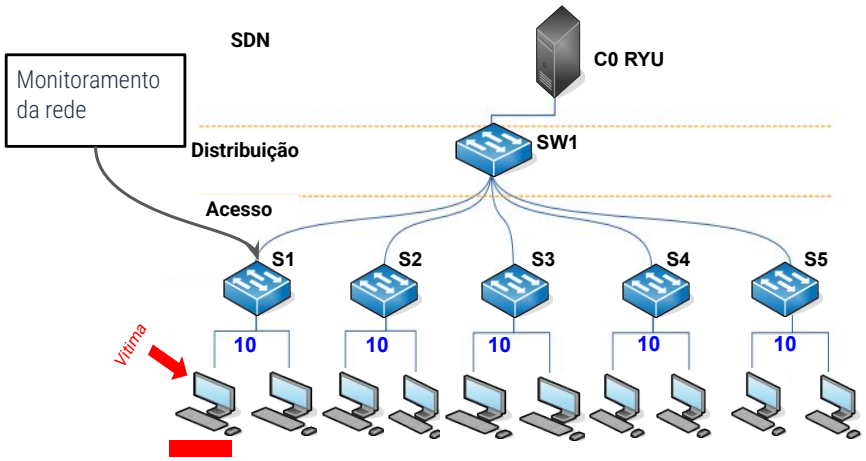

Figura 5. Rede SDN

O modelo *Transformer* utilizado nesta avaliação foi configurado com *learning rate* de 0,0001 e *chunks* de 120 pacotes. O critério de rotulagem adotado para essas janelas estabelece que, caso qualquer pacote dentro do *chunks* seja associado a um endereço malicioso, toda a janela é classificada como ataque. Essa estratégia garante que o modelo possa ser treinado usando dados rotulados para aprender a identificar padrões maliciosos mesmo quando eles estão dispersos em uma sequência maior de tráfego benigno. A ferramenta utilizada para a coleta do tráfego de foi o *tshark* (Seção 3.3.2). O *tshark* permite extrair diretamente os atributos requeridas para a construção das janelas de análise. Parte dos atributos refletem as características da rede. Esses atributos são a média de pacotes e tamanho médio de pacotes em uma janela temporal, que não podem ser obtidos diretamente, são realizados cálculos complementares para reconstrução manual dos campos faltantes. Essa estratégia possibilita que todo o fluxo de dados seja mantido em memória, eliminando a necessidade de operações de escrita em disco e acelerando significativamente a etapa de captura.

4.2. Experimentos

Este trabalho apresenta os resultados de três experimentos para avaliar o método proposto. Os **experimentos 1 e 2**, denominados de **Exp. 1** e **Exp. 2**, correspondem a protocolos

diferentes, sendo eles SYN e UDP respectivamente. O **experimento 3** (**Exp. 3**) apresenta um ataque com dois protocolos diferentes juntos, SYN e UDP. Todos os experimentos usam a mesma arquitetura SDN apresentada na subseção anterior, mudando os dispositivos selecionados para os ataques. Todos os experimentos usam o mesmo *Transformer* com o mesmo *learning rate*, número de épocas e tamanho de *chunks*. O que os distingue, principalmente, são os atributos selecionados em cada caso.

Os atributos utilizados no Exp. 1, ataque SYN *flood*, se baseiam na coleta de informações do cabeçalho TCP, que são número de sequência, número de confirmação, conjunto de *flags* do TCP, *flag* SYN, *flag* ACK, *flag* RST, *flag* FIN, *flag* PSH, *flag* URG, *flag* ECE, *flag* CWR, tamanho da janela, tamanho do cabeçalho TCP, tamanho do *payload* TCP, valor do MSS, fator de escala da janela e indicação de suporte a SACK. A diferença entre esses experimentos está relacionada ao software empregado para sua execução. No ataque UDP, Exp. 2, os atributos são extraídos de forma análoga ao Exp. 1, considerando-se as informações disponíveis no próprio cabeçalho UDP, sendo elas comprimento do *datagrama* UDP, *checksum* UDP e tamanho do *payload* UDP. O Exp. 3 combina tráfego UDP e TCP e todos os atributos mencionados para os experimento 1 e 3 são combinados no mesmo arquivo. Os campos ausentes do protocolo TCP são preenchidos com zero quando o pacote corresponde ao protocolo UDP. De modo equivalente, quando o pacote é TCP, os campos específicos de UDP são preenchidos com zero. Dessa forma, o modelo é capaz de aprender as características tanto dos protocolos TCP e UDP, como as características de fluxo de pacote da rede, os quais são utilizados independentemente do protocolo empregado. No modo de teste e análise de tráfego de rede, é extraído o timestamp da rede para o cálculo das janelas temporais. Por fim para quantificar as métricas de avaliação (matriz de confusão, recall, f1-score, acurácia e precisão) na etapa de análise de tráfego de rede, é preciso saber quais os endereços de IP são provenientes de dispositivos infectados e quais dispositivos tem trafego legítimos.

Tabela 1. Tabela de Métricas

Experimento	F1-score	Precisão	Acurácia	Recall	FN	FP	VP	VN
Exp. 1	99%	99%	99%	100%	8	2176	1483853	41896
Exp. 2	100%	100%	100%	100%	1	0	268145	45789
Exp. 3	99%	99%	99%	100%	1	2691	1141426	46274

Os resultados apresentados na Tabela 1 representam os resultados dos experimentos citados acima. Cinco dispositivos são selecionados para produzir tráfego benigno por um minuto, e 25 dispositivos para realizar o ataque. Os dispositivos de ataque iniciam todos ao mesmo tempo, atacando o servidor durante um minuto. Esses tempos são misturados, começando com 30 segundos de tráfego estritamente benigno, 30 segundos com os dois tráfegos simultaneamente e, por fim, 30 segundos somente com tráfego malicioso. Desta forma, a Tabela 1 representa os resultados de todos os ataques usados no experimento usando as métricas: *F1-Score*, Precisão, Acurácia, *Recall*, falsos negativos (FN), falsos positivos (FP), verdadeiro positivo (VP) e verdadeiro negativo (VN).

A Tabela 2 mostra o desempenho em tempo de execução, que inclui o tempo total de pré-processamento e inferência do modelo. Além disso, avaliou-se como a quantidade de *chunks* processadas por janela influencia o tempo de detecção.

Tabela 2. Tabela de Tempos de Inferência da Janela por chunks

Número de chunks	Tempo Benigno (ms)	Tempo Ataque(ms)
1	1200	7
50	160	35
372	150	150

4.3. Discussão

As métricas apresentadas na Tabela 1 se mantiveram acima de 99% em todos os experimentos utilizados, destacando a capacidade do modelo para conseguir identificar ataques presentes nas *chunks*. As métricas FN, FP, VP e VN representam a matriz de confusão. O modelo acertou grande parte das *chunks* analisadas, tendo grande parte dos erros concentrados nos falsos positivos, resultado esperado pela lógica de execução do *Transformer*, que se baseia em qualquer pacote de rede proveniente de algum dispositivo infectado para caracterizar a *chunk* toda como ataque, isso faz com que o *Transformer* classifique erroneamente certas *chunks* que tenham o mínimo de comportamento suspeito. Mas ao mesmo tempo garante que o *Transformer* consiga detectar ataques em tráfegos predominantemente benignos. Para contornar esse problema, trabalhos futuros podem considerar análises complementares apenas nas *chunks* que foram identificadas com ataques. Assim, seria possível lidar com a possibilidade de falsos positivos.

Em relação ao tempo, o desempenho do modelo vária conforme a quantidade de *chunks* utilizadas para a detecção, como mostrado na Tabela 2. As métricas de desempenho do modelo presente na Tabela 1 se mantém independe da quantidade de *chunks* processadas. A diferença do tempo apresentada na Tabela 2 ocorre porque o consumo da janela de cinco segundos afeta diretamente o tempo total de processamento. Quando a janela é consumida *chunk* a *chunk*, a detecção tende a ser mais rápida, pois a execução pode ser interrompida assim que o ataque é identificado.

Em contrapartida, quando o modelo precisa varrer a janela inteira, o tempo aumenta. Quando a janela é consumida de uma vez só, representada pelas 372 *chunks*, média de tamanho das janelas completas neste experimento, as janelas com apenas tráfego benigno tendem a ser processadas mais rapidamente. Entretanto, janelas com ataque podem apresentar tempo maior do que o necessário, pois seria possível interromper a execução antes e ainda assim obter a resposta. Os melhores resultados foram obtidos com o processamento de 50 *chunks* em paralelo, o tempo para processar a janela completa não cresce tanto quanto no processamento integral, e a detecção se mantém rápida quando o ataque ocorre nas primeiras 50 *chunks*, preservando a eficiência mesmo quando é necessário percorrer toda a janela. O fato de as *chunks* estarem sendo processadas em paralelo na GPU desempenhou um papel significativo na obtenção dos tempos deste experimento sendo usada em 100%, hardwares mais potentes tenderiam a reduzir ainda mais esses tempos.

Outra discussão válida diz respeito ao aumento na quantidade de dispositivos e, consequentemente, ao crescimento do volume de tráfego. Quanto maior o número de *chunks* em uma janela, maior será o tempo de inferência do modelo, uma vez que será necessário percorrer um número superior de *chunks* para concluir o processamento de uma janela. Para mitigar esse problema, algumas estratégias podem ser adotadas. Por exemplo, o aumento do tamanho unitário dos *chunks*, de modo que cada janela contenha

menos elementos, ou a redução dos intervalos das janelas temporais, abordagem que, embora preserve os tempos de inferência, implica maior perda de contexto.

4.4. Comparação

Para a comparação, foi construído um outro *Transformer* mais complexo, tendo como configuração 3 *encoders* e 3 *decoders*, cada um possuindo 4 camadas de atenção, dimensão de 64, 0,1 de *layer dropout* e uma função de ativação GELU, que pondera as entradas pela sua magnitude, com um *threshold* de 90%. Esse *Transformer* foi treinado com o mesmo *dataset* que o Exp.3, logo, foi treinado para detectar ataques UDP e SYN *flood*. Para a etapa de avaliação contínua na rede, foi feita outra captura da mesma forma da Seção 4.2, sendo 1 minuto e 30 segundos de captura total, 30 segundos de tráfego benigno, 30 segundos de tráfego benigno e de ataque e 30 segundos somente de ataque.

Na avaliação pós-treino, os 2 modelos performaram praticamente iguais, mantendo 99% nas métricas de *f1-score*, *recall*, acurácia e precisão. Apesar de o *Transformer* mais complexo ter previsão de levar mais tempo na inferência das janelas, na avaliação contínua de rede os modelos também mantiveram os mesmos tempos, com algumas dezenas de milissegundos de diferença no valor médio por janela. Isso se deve à classificação errônea das janelas benignas pelo *Transformer* mais complexo, parando na primeira *chunk* em todas as janelas benignas, diferentemente do *Transformer* do Exp.3, que classificou todas as janelas, benignas ou não, de forma correta.

5. Conclusão

Diante do crescimento da frequência e da rapidez dos ataques DDoS, é de extrema importância realizar a detecção desses ataques de maneira assertiva e com baixa latência. Os modelos de IA, apesar de apresentarem ótimas taxas de acerto e capacidade de analisar tanto o tráfego de rede quanto os pacotes, possuem um custo computacional muito elevado, inviabilizando a detecção em tempo de execução. Para lidar com esse problema, este artigo propôs uma abordagem em *chunks* juntamente com um modelo *Transformer* otimizado, a fim de auxiliar na detecção mais rápida, auxiliada com a execução em GPU. Com os experimentos realizados, é possível concluir que o modelo *Transformer* baseado em *chunks* pode ser aplicado à detecção de ataques e resulta em baixa latência. Esses tempos ainda podem ser otimizados, uma vez que as *chunks* são agregadas integralmente antes do processamento. Caso as *chunks* sejam processadas à medida que alcancem a quantidade definida, o tempo total, tanto para janelas benignas quanto para janelas com ataque, tende a diminuir, evidenciando o potencial do modelo para uso em serviços críticos que exigem detecção rápida. Como trabalhos futuros, o modelo pode ser treinado em outros cenários, com diferentes tipos de ataque e lógicas diferentes de classificação para lidar melhor com trafego legítimo que esta intercalado com ataques, de forma a lidar apenas com os pacotes maliciosos. Além de ser implantado em um servidor com GPU dedicada e maior poder computacional, visando reduzir ainda mais o tempo de execução, assim como uma nova bateria de testes em uma rede real, com um servidor como alvo, empregando o modelo proposto para a detecção de ataques.

Referências

Ali, M., Saleem, Y., Hina, S., and Shah, G. A. (2025). DDoSViT: IoT DDoS attack detection for fortifying firmware Over-The-Air (OTA) updates using vision transformer. *Internet of Things*, 30:101527.

Ashraf, J., Raza, G. M., Kim, B.-S., Wahid, A., and Kim, H.-Y. (2025). Making a real-time iot network intrusion-detection system (inids) using a realistic BoT-IoT dataset with multiple machine-learning classifiers. *Applied Sciences (2076-3417)*, 15(4).

Dantas Silva, F. S., Silva, E., Neto, E. P., Lemos, M., Venancio Neto, A. J., and Esposito, F. (2020). A taxonomy of DDoS attack mitigation approaches featured by SDN technologies in iot scenarios. *Sensors*, 20(11):3078.

Farias, E. P., de Neira, A. B., Borges, L. F., and Nogueira, M. (2025). Transformers model for DDoS attack detection: A survey. *Computer Networks*, 270:111433.

Gcore (2025). Gcore successfully stops 6 tbps DDoS attack. Pico de 6 Tbps e 5,3 Bpps; ataque UDP; mitigado.

He, M., Zhao, X., and Wang, X. (2024). An efficient ddos detection method based on packet grouping via online data flow processing. *IEEE Transactions on Sustainable Computing*, 10(2):202–216.

Hernandez, D. V., Lai, Y.-K., and Ignatius, H. T. (2025). Real-time DDoS detection in high-speed networks: A deep learning approach with multivariate time series. *Electronics*, 14(13):2673.

Holdsworth, J. and Kosinski, M. (n.d.). *O que é um ataque distributed denial-of-service (DDoS)?* IBM Think.

Li, Y., Deng, X., Yang, A., and Gao, J. (2025). A transformer-based framework for DDoS attack detection via temporal dependency and behavioral pattern modeling. *Algorithms*, 18(10):628.

Mittal, M., Kumar, K., and Behal, S. (2023). Deep learning approaches for detecting DDoS attacks: A systematic review. *Soft computing*, 27(18):13039–13075.

Najar, A. A. and Manohar Naik, S. (2025). Ddos attack detection using cnn-bilstm with attention mechanism. *Telematics and Informatics Reports*, 18:100211.

Nalayini, C., Soumya, T., Lalitha, S., and Tamijetchelvy, R. (2025). A novel adaptive transformer based quantum intrusion detection system for software defined networks. *Scientific Reports*, 15(1):36505.

of the Republic of Lithuania, G. (2022). Intense DDoS attacks targeted several companies and institutions in lithuania.

Ullah, F., Ullah, S., Srivastava, G., and Lin, J. C.-W. (2024). IDS-INT: Intrusion detection system using transformer-based transfer learning for imbalanced network traffic. *Digital Communications and Networks*, 10(1):190–204.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Wang, H. and Li, W. (2021). DDosTC: A transformer-based network attack detection hybrid mechanism in SDN. *Sensors*, 21(15):5047.

Wireshark (2026). *tshark - The Wireshark Network Analyzer 3.2.2*. Online documentation.