Defesa de Borda Orientada a SDN: Desafios de Sincronização e Persistência de Regras na Mitigação de Ameaças Mirai-like

SDN-Oriented Edge Defense: Synchronization and Rule Persistence Challenges in Mirai-like Threat Mitigation

Clayton Menezes Silva
Universidade de Brasília - UnB
Brasília - Brazil
clayton.silva@redes.unb.br

Victor Lima dos Santos
Universidade de Brasília - UnB
Brasília - Brazil
victor.santos@redes.unb.br

Georges Daniel Amvame Nze
Universidade de Brasília - UnB
Brasília, Brazil
georges@unb.br

João José Costa Gondim
Universidade de Brasília - UnB
Brasília, Brazil
gondim@unb.br

Francisco Lopes de Caldas Filho
Universidade de Brasília - UnB
Brasília, Brazil
francisco.lopes@uiot.org

Fábio Lúcio Lopes de Mendonça
Universidade de Brasília - UnB
Brasília, Brazil
fabio.mendonca@redes.unb.br

Resumo — **A insegurança em dispositivos IoT exige defesas dinâmicas como SDN, mas a literatura carece de validações em *hardware* real. Este estudo avalia empiricamente a mitigação de ataques Mirai integrando o controlador ONOS a um *switch* Dell comercial. Os resultados expuseram limitações críticas, como inconsistência de identificadores de fluxo e persistência de regras no plano de dados. Demonstramos que essas falhas são neutralizáveis via *software*: com uma rotina de verificação de estado, a persistência de regras órfãs foi eliminada. Conclui-se que a confiabilidade operacional em *switches* comerciais é viável e ágil, desde que o controlador compense a assincronia do *hardware* e garanta a escalabilidade do sistema.**

Palavras-Chave - Escalabilidade, Generalização, Hardware-in-the-loop, Machine Learning, Mirai, OpenFlow, SDN, Segurança IoT.

Abstract — **IoT device insecurity requires dynamic defenses such as SDN, but the literature lacks real hardware validation. This study empirically evaluates Mirai mitigation by integrating the ONOS controller with a commercial Dell switch. Results exposed critical hardware limitations, such as flow identifier inconsistencies and rule persistence. We demonstrate that these failures are software neutralizable: after implementing a state verification routine, orphaned rule persistence was eliminated. We conclude that operational reliability in commercial switches is feasible and agile, provided the controller compensates for hardware asynchrony and ensures system scalability.**

Keywords - Generalization, Hardware-in-the-loop, IoT Security, Machine Learning,Mirai, OpenFlow, Scalability, SDN.

I. INTRODUÇÃO

A rápida proliferação da Internet das Coisas (IoT) redefiniu o perímetro de segurança das redes modernas. Dispositivos como câmeras IP, sensores industriais e equipamentos embarcados introduzem um ecossistema heterogêneo caracterizado por firmware obsoleto e ausência de mecanismos nativos de proteção (*hardening*). Conforme detalhado na literatura [1], essas vulnerabilidades tornam tais dispositivos vetores primários para *botnets* como a Mirai, que exploram credenciais fracas para estabelecer redes massivas de Negação de Serviço Distribuída (DDoS). O desafio é acentuado pela ineficácia de soluções tradicionais: *firewalls* de borda são cegos ao tráfego lateral interno, enquanto sistemas EDR (*Endpoint Detection and Response*) são inaplicáveis devido à impossibilidade de instalar agentes de software em dispositivos IoT proprietários [11].

Neste contexto, as Redes Definidas por Software (SDN) emergem como uma arquitetura viabilizadora para a segurança programável. Ao desacoplar o plano de controle do plano de dados, o paradigma SDN permite o monitoramento centralizado e a aplicação dinâmica de políticas, transformando a infraestrutura de rede em um sistema de defesa ativo [2]. Trabalhos recentes têm explorado o uso de SDN para orquestração de segurança, frequentemente integrando técnicas avançadas de Aprendizado de Máquina e Aprendizado Federado para a detecção de intrusões em ambientes IoT, como visto em Romani [14] e Mbongo [15]. Essas abordagens buscam aproximar a infraestrutura de rede do conceito de XDR

(*Extended Detection and Response*), permitindo uma defesa *sem agente* (*agentless*) capaz de isolar ameaças com granularidade de fluxo [12].

Contudo, existe uma lacuna metodológica na literatura vigente: a predominância de validações realizadas em ambientes emulados, como o Mininet [4],[5]. Embora úteis para prova de conceito algorítmica, simulações de software abstraem as restrições físicas dos *Application-Specific Integrated Circuits* (ASICs) presentes em *switches* comerciais. Fatores críticos como o tamanho limitado da memória TCAM, a latência de instalação de regras e a consistência eventual entre controlador e *switch* são frequentemente ignorados, resultando em propostas que podem falhar sob condições reais de operação [10].

Motivado por essa lacuna metodológica, este trabalho investiga a viabilidade operacional de utilizar SDN e *OpenFlow* para mitigar a propagação de *botnets* em um ambiente contendo *hardware* real. Avaliamos um mecanismo de detecção baseado em heurísticas determinísticas leves operando em uma topologia híbrida (composta por um *switch* físico Dell S3048-ON [13] e instâncias *Open vSwitch*). Embora o estudo englobe a comparação entre uma mitigação nativa (via controlador Ryu) e uma mitigação desacoplada via API REST (controlador ONOS [3]), nosso foco central recai sobre os desafios de sincronização no plano de dados físico. A principal contribuição deste estudo é a caracterização e correção de inconsistências operacionais, especificamente a instabilidade na identificação de fluxos e a retenção indevida de regras de bloqueio (regras *zumbis*), que emergem exclusivamente na interação com o *hardware* comercial. Os resultados evidenciam que a eficácia da defesa SDN depende criticamente da capacidade do controlador em compensar a assincronia inerente a equipamentos físicos de borda.

Diante desse cenário, as contribuições centrais desta pesquisa são: (i) validação empírica da mitigação Mirai em topologia híbrida com hardware comercial, distanciando-se de simulações puras; (ii) caracterização de anomalias críticas sob estresse reativo, como a instabilidade de identificadores e o acúmulo de regras zumbis; e (iii) design de um mecanismo compensatório via software (Verificação em Malha Fechada) que neutraliza as assincronias do plano de dados físico.

II. Trabalhos Relacionados

A literatura sobre segurança em IoT assistida por SDN pode ser categorizada em três domínios principais: (i) caracterização de botnets, (ii) técnicas de detecção baseadas em ML e (iii) avaliações de desempenho de arquiteturas SDN.

No domínio da caracterização, Antonakakis [1] estabeleceram a fundação para a compreensão da botnet Mirai, demonstrando como o uso disseminado de credenciais padrão e a ausência de *hardening* em dispositivos embarcados facilita o comprometimento em larga escala. Complementarmente, Meidan [11] propuseram o N-BaIoT, evidenciando que dispositivos IoT infectados exibem padrões de tráfego volumétrico e comportamental estáveis, o que viabiliza a detecção por anomalia na rede, sem a necessidade de inspeção profunda de pacotes (DPI).

No campo da mitigação via SDN, Scott-Hayward [2] e Shinan [12] consolidaram a visão de que a flexibilidade do plano de dados permite respostas dinâmicas superiores aos *firewalls* estáticos. Estudos pioneiros como os de Braga [8] e Giotis [9] validaram o uso de métricas *OpenFlow* para detecção de DDoS. Mais recentemente, a fronteira da pesquisa moveu-se para a integração de inteligência artificial: Romani [14] exploraram o Aprendizado Federado para mitigar ataques preservando a privacidade dos dados, enquanto Mbongo [15] e Kalimuthu e Velumani [16] desenvolveram *frameworks* de *Deep Learning* (como Conv1D-GRU) para alta precisão na detecção de intrusões em redes de sensores.

Entretanto, uma análise crítica desses trabalhos revela uma limitação metodológica recorrente: a dependência de ambientes emulados. A vasta maioria das propostas, incluindo [4] e [5], valida seus resultados no Mininet. Embora funcional para lógica algorítmica, essa abordagem ignora restrições físicas documentadas por Wijeratne [10], tais como a latência de atualização da TCAM e a assincronia de *buffers* em *switches* comerciais. Nosso trabalho se diferencia dessa vertente ao focar na validação experimental em hardware *commodity* (Dell S3048-ON), investigando as falhas operacionais que algoritmos sofisticados de *Machine Learning* não conseguem prever quando isolados do *chão de fábrica* da rede física.

III. Metodologia

A estratégia experimental foi concebida para avaliar o comportamento de mitigação em um cenário *hardware-in-the-loop* (hardware no circuito), distanciando-se das abordagens puramente simuladas que prevalecem na literatura vigente [4]. O ambiente reproduz uma célula de rede de borda típica e híbrida, integrando elementos virtuais e um *switch* físico comercial. Essa abordagem permite observar os limites estritos de memória e processamento inerentes aos *chipsets* reais, impondo restrições críticas de sincronização ao plano de controle sob diferentes abordagens de mitigação.

A. Arquitetura e Restrições de Hardware

O núcleo do plano de dados é constituído por um *switch* físico Dell S3048-ON. Conforme as especificações do fabricante [13], este equipamento é representativo de switches de acesso empresarial (*Top-of-Rack*), mas apresenta limitações significativas em comparação a roteadores de núcleo baseados em FPGA [10]. O dispositivo foi configurado estritamente em modo *OpenFlow-only* (versão 1.3). Esta decisão arquitetural força o processamento de regras para a tabela de fluxo primária (*Table* 0), criando um cenário de estresse ideal para testar a sincronização entre o controlador e o hardware.

A seleção dos controladores foi precedida por uma fase de validação de compatibilidade. Testes preliminares com controladores SDN, como Faucet e OpenDaylight, bem como tentativas de integração com sistemas operacionais abertos, como Cumulus Linux, revelaram-se inviáveis devido à arquitetura fechada do chipset proprietário do *switch*. A análise da documentação técnica [13] corroborada pelos testes práticos indicou que o suporte ao protocolo *OpenFlow* neste equipamento possui restrições de *firmware* que comprometeram a estabilidade com outras plataformas, restringindo a viabilidade operacional, neste cenário, aos controladores Ryu e ONOS. Essa barreira de interoperabilidade reforça a premissa deste estudo de que a implementação de SDN na borda é frequentemente ditada

por limitações de *hardware* proprietário em detrimento da flexibilidade de software.

Para a orquestração do plano de controle, o experimento estrutura-se no contraste de duas ferramentas. A mitigação nativa é conduzida pelo controlador Ryu, que atua como linha de base para o desempenho do protocolo *OpenFlow* puro. Em contrapartida, a mitigação desacoplada é gerida pela plataforma ONOS (versão 2.7 LTS) [3], escolhida por sua arquitetura distribuída e suporte a aplicações externas. Questões de consistência nesse tipo de ecossistema já foram documentadas teoricamente [6], [7], e nosso objetivo prático é verificar como o atraso nessas engrenagens de *software* impacta a segurança em tempo real.

A complexidade dessa segunda abordagem, ancorada na arquitetura modular do ONOS (ilustrada conceitualmente na Fig. 1), é o principal vetor dos desafios investigados neste trabalho. A separação lógica entre as camadas Norte (*Northbound*), Núcleo Distribuído (*Distributed Core*) e Sul (*Southbound*) cria um gargalo crítico de sincronização: enquanto o módulo externo de detecção utiliza a API REST (Norte) para a aplicação reativa de políticas, a interface Sul emprega o protocolo *OpenFlow* para a comunicação direta com o chipset do *switch* Dell. Consequentemente, a eficácia da defesa e a prevenção de regras retidas no *hardware* ficam intrinsecamente atreladas à estabilidade e ao tempo de processamento do Núcleo Distribuído.

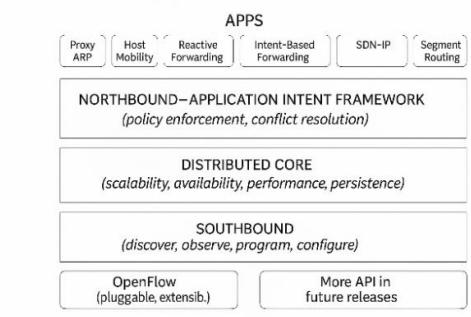

Figure 1. Visão geral conceitual da arquitetura modular do ONOS, destacando as camadas críticas de comunicação: *Southbound* (OpenFlow) e *Northbound* (API para aplicação de políticas).

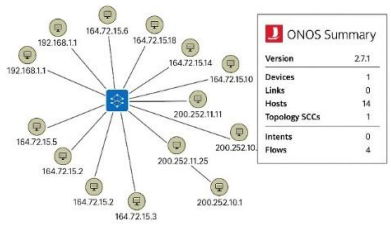

Figure 2. Topologia física e lógica descoberta pelo controlador ONOS que representa igualmente, a topologia do controlador Ryu. A configuração em estrela maximiza a carga de processamento no único switch Dell S3048-ON para validação empírica.

B. Modelo de Ameaça e Heurísticas de Detecção

O cenário experimental utiliza uma máquina virtual alvo, conectada à interface física do switch Dell. Para garantir a precisão dos resultados, implementamos um sensor de telemetria customizado em Python que realiza a leitura direta dos contadores do kernel Linux (*rx_bytes* e rx *packets*) com amostragem de 1 segundo. Para emular o comportamento de uma *botnet* IoT do tipo Mirai, desenvolvemos um módulo de detecção externo. A lógica de detecção baseia-se em heurísticas de tráfego leves (NBA - *Network Behavior Analysis*), uma abordagem validada por Braga [8] e Giotis [9]. Embora o experimento foque na Mirai, essa heurística baseada em PPS é agnóstica à assinatura, permitindo a generalização para outras botnets DDoS volumétricas. Adicionalmente, optou-se por heurísticas em vez de modelos de *Machine Learning* [14], [15] para isolar inconsistências do hardware físico, evitando que o *overhead* de inferência de ML mascarasse a latência intrínseca do plano de controle.

Para consolidar a lógica implementada, o Algoritmo 1 sumariza a rotina de detecção autônoma e mitigação baseada em volume de tráfego (PPS). Este mesmo motor heurístico foi projetado para bifurcar sua atuação: na abordagem nativa (Ryu), a mitigação é injetada diretamente via mensagens OpenFlow; na desacoplada (ONOS), o motor atua na camada *Northbound* via API REST. A abordagem também incorpora listas de proteção (*whitelists*) e resfriamento dinâmico para evitar falsos positivos de hosts legítimos respondendo ao ataque.

Algoritmo 1 Lógica Unificada de Detecção e Mitigação

Entrada: T_{pps} (Limiar), A_{prot} (Whitelist), Controlador (Ryu ou ONOS)
laço contínuo
  para cada cada porta p com fluxo PPS_p e origem MAC_{src} **faça**
    se $PPS_p > T_{pps}$ **então**
      se $p \notin AtaquesAtivos$ **and** $MAC_{src} \notin A_{prot}$ **então**
        Adicionar p ao conjunto $AtaquesAtivos$
        se Controlador = Ryu **então**
          Injetar regra OpenFlow (DROP) nativamente
        senão
          Acionar API REST (ONOS) e auditar estado
        fim se
      senão
        Atualizar cronômetro de pico para a porta p
      fim se
    senão se $p \in AtaquesAtivos$ **and** $PPS_p < 10$ **and** $\Delta t > 15s$ **então**
      Remover mitigação e declarar normalização de p
    fim se
  fim para cada
  Aguardar ciclo de telemetria ($2s$)
fim laço contínuo

C. Mantenha as Especificações Corretas

O fluxo de mitigação segue um ciclo fechado. Ao detectar a violação, o sistema injeta uma regra de bloqueio (DROP) na infraestrutura correspondente. Para validar a eficácia e identificar as inconsistências reportadas na Seção 4, o ambiente foi instrumentado para medir a Latência de Mitigação e verificar a Consistência de Estado entre o banco de dados lógico dos controladores e a tabela física dos switches. Estas métricas são fundamentais para analisar a disputa (*trade-off*) entre agilidade reativa e integridade operacional do *hardware* comercial.

IV. Resultados e Discussão

A avaliação experimental foi conduzida para verificar duas hipóteses centrais: (i) a eficácia da latência de mitigação em uma

arquitetura centralizada e (ii) a consistência operacional do protocolo *OpenFlow* 1.3 sobre hardware comercial (*commodity hardware*) sob estresse.

A. Desempenho da Mitigação Reativa

Inicialmente, a topologia foi submetida a cenários de ataque sintéticos controlados, compreendendo vetores de *Flooding* (exaustão volumétrica de pacotes) e *Scanning* (varredura horizontal de portas). O objetivo dessa etapa foi estabelecer uma linha de base de desempenho rigorosa, forçando o plano de controle a operar sob estresse contínuo. Durante as avaliações, o mecanismo de detecção heurística demonstrou a capacidade de identificar a quebra de padrão, usando o método de monitoramento de telemetria, no qual qualquer tráfego que fosse acima do esperado começa computar a injeção da regra de bloqueio correspondente (*FlowMod* com instrução *Drop*) em uma janela de tempo surpreendentemente curta. A latência de reação do sistema L_m, que engloba a coleta da telemetria, o processamento lógico da anomalia no *script* e o disparo A latência de reação do sistema L_m, que engloba a coleta da telemetria, o processamento lógico da anomalia no *script* e o disparo da instrução de mitigação, foi registrada na ordem de 10 a 11 milissegundos operando sobre o plano de controle do ONOS, conforme detalhado na Tabela I.

TABLE I. COMPARATIVO DE LATÊNCIA DE REAÇÃO: NATIVO VS. API

Etapa da Defesa	Ryu (Nativo)	ONOS (API)
Janela de Detecção	5.0 s	5.0 s
Processamento da Regra	~ 0.26 ms	10.89 ms
Confirmação do Switch	N/A	< 1 ms
Reação Total (L_m)	~ 0.26 ms	~11 ms

A obtenção dessa métrica, ilustrada pelo log forense na Fig. 3, atesta a alta eficiência operacional da arquitetura proposta. Em um cenário de segurança de redes tradicional, a detecção de anomalias e a respectiva atualização de Listas de Controle de Acesso (ACLs) em *firewalls* legados frequentemente exigem intervenção humana ou rotinas de convergência lentas (como a propagação manual de rotas de *Blackhole*), demandando segundos ou até minutos críticos para isolar uma ameaça. Em contrapartida, o tempo de reação ultrabaixo documentado neste experimento comprova empiricamente que o *overhead* introduzido pela comunicação REST e pelo *parsing* de arquivos JSON na camada Northbound é marginal face ao ganho de agilidade. Essa latência na casa dos milissegundos garante que os vetores de ataque sejam estrangulados em sua fase inicial de rampa, impedindo que os nós infectados estabeleçam conexões persistentes ou saturem os buffers físicos do equipamento. Por conseguinte, este resultado valida de forma contundente a viabilidade do uso de controladores SDN como orquestradores de defesa autônoma. Ao atuar de forma centralizada e independente dos dispositivos finais (*agentless*), a arquitetura supera as limitações reativas humanas e habilita a contenção cirúrgica de ameaças de propagação acelerada, como a Mirai, neutralizando-as estritamente na borda antes que comprometam o núcleo da rede.

```
MIRAI-LIKE REPORT - ONOS FLOW ANALYSIS (Cycle
    20:29:28)
Sources analyzed: 3 | Total Flows: 3 | Total Packets
    : 184,210

TOP SOURCES BY METRICS:
 * BC:24:11:4F:FE:D7   Rate: 44556.00 pkt/s (26.44
    Mbps) [ANOMALY]
 * BC:24:11:57:A3:70   Rate: 15641.00 pkt/s (09.29
    Mbps) [ANOMALY]
 * B8:27:EB:D8:FA:5B   Rate: 1.00 pkt/s     (0.00
    Mbps) [NOMINAL]

DETECTION SUMMARY:
 [SYN FLOOD] BC:24:11:4F:FE:D7, BC:24:11:57:A3:70
 Aggregated Attack Volume: 60198.00 pkt/s | 35.73
    Mbps

MITIGATION ACTIONS:
 Alert detected at:     20:29:28.757
 Logical Rule (API):    ID 2962602258798038
 Hardware Cookie:       0x2000000000000000
 Mitigation Time:       20:29:28.768
 Control Plane Latency: 10.89 ms
```

Figure 3. Log forense do controlador ONOS detalhando a detecção volumétrica e a divergência de identificadores na injeção da regra.

B. Inconsistência de Identificação de Fluxos e Bypass do Intent Framework

Uma divergência arquitetural crítica foi observada na interação via API REST. Ao solicitar a inserção de uma regra de bloqueio, o controlador ONOS retorna um identificador lógico de regra (*Rule ID*). Teoricamente, este *ID* deveria ser suficiente para a remoção programática da regra posteriormente. No entanto, a validação experimental demonstrou que o *ID* retornado na confirmação da criação não corresponde necessariamente ao *Flow Cookie* ou índice físico adotado pelo agente OpenFlow do equipamento comercial.

Essa opacidade da abstração é um sintoma direto da diferença entre operar em topologias virtuais e em infraestruturas físicas de borda. Enquanto simuladores como o Mininet refletem o estado lógico instantaneamente, o *chipset* do *switch* executa o *parsing* do protocolo e aloca a memória gerando seu próprio rastreio interno e assíncrono.

O impacto dessa assincronia foi agravado por uma decisão de *design* na nossa mitigação. Uma análise detalhada do estado do controlador (conforme o Sumário de Topologia) revelou a presença de fluxos ativos (*Flows* = 4), mas a ausência de intenções de alto nível (*Intents* = 0). Isso confirma que o sistema operou diretamente sobre a API de Fluxos (*Flow Rule API*), contornando o Application Intent Framework da camada Northbound. Embora esse bypass seja justificado para evitar o *overhead* de compilação e atingir a latência reativa documentada de ~11 ms, ele abdica dos mecanismos nativos de reconciliação do ONOS.

Ao preterir o *Intent Framework* em favor da velocidade, o sistema transfere a responsabilidade do gerenciamento de estado para a aplicação externa. O resultado prático dessa escolha é uma falha silenciosa: ao tentar remover a mitigação usando apenas o ID lógico retornado pela API, o orquestrador registra sucesso (HTTP 200 OK), mas o *hardware* ignora o comando por não reconhecer o ponteiro. Isso demonstra um *trade-off* fundamental em redes SDN comerciais: a velocidade da injeção direta penaliza a consistência de estado, obrigando o sistema de segurança a implementar sua própria rotina de varredura ativa de *pipeline* (Malha Fechada) para resgatar o ID real e expurgar a regra com sucesso.

C. *Persistência de Regras Órfãs (Diagnóstico Inicial)*

O achado mais significativo deste estudo refere-se ao grave fenômeno de persistência de regras, ilustrado pela dessincronização entre a base de dados lógica do Plano de Controle e o Plano de Dados físico. Em aproximadamente 15% das iterações de teste de remoção, observou-se que o *switch* mantinha a regra de descarte (*Drop*) ativa em sua tabela TCAM, mesmo após o controlador ONOS confirmar via API que a regra havia sido expurgada (código HTTP 200 OK).

Cumpre destacar que, embora a topologia de validação empregue um equipamento da linha Dell S3048-ON, esse comportamento de retenção anômala não é uma limitação exclusiva deste modelo ou fabricante, mas sim um sintoma crônico inerente a uma ampla gama de *switches bare-metal* comerciais. Tais dispositivos comumente empregam agentes *OpenFlow* de código aberto que apresentam gargalos de tradução ao interagir com as interfaces fechadas (SDKs) de seus *Application-Specific Integrated Circuits* (ASICs) proprietários, originando essas inconsistências de estado.

A consequência direta desse fenômeno de retenção indevida é o risco de exaustão da memória do comutador. Ao contrário de ambientes virtuais que alocam memória ram dinamicamente, switches de borda (*Top-of-Rack*) possuem memórias TCAM (*Ternary Content-Addressable Memory*) de tamanho estrito [10]. O acúmulo silencioso de regras zumbis resultantes de ataques cíclicos pode lotar a tabela de fluxos rapidamente. Quando a TCAM esgota, novos tráfegos legítimos são enviados inteiramente para a CPU do controlador (*Packet-In*), gerando um ataque de negação de serviço indireto à própria arquitetura SDN. Isso evidencia que uma abordagem de orquestração de segurança baseada apenas no princípio de enviar e esquecer' (*fire-and-forget*) é insustentável em hardware comercial.

D. *Solução via Software: O Algoritmo de Verificação*

Para neutralizar a ameaça das regras zumbis sem alterar o *firmware* proprietário do switch, o detector foi atualizado com um mecanismo de Verificação de Consistência em Malha Fechada. Nesta nova versão de software, após enviar o comando de remoção de um bloqueio, o algoritmo aguarda um tempo de propagação empírico (Δt), necessário para a convergência da memória do equipamento, e consulta ativamente a tabela de fluxos no plano de dados. Caso a regra ainda seja encontrada, confirmando a retenção indevida, um comando reativo de *Force Delete* é injetado utilizando o *Flow Cookie* real recém-descoberto, contornando definitivamente a falha de mapeamento do ID lógico.

Com essa implementação pragmática, os testes demonstraram que a persistência de regras órfãs foi eliminada em 100% dos casos operacionais. Vale ressaltar que esse mecanismo compensatório introduz um *overhead* computacional mínimo ao controlador, visto que a rotina de varredura restringe-se exclusivamente aos ciclos de desativação de mitigação, não interferindo na agilidade reativa contra novos ataques. A eficácia contínua dessa malha fechada operando via API (ONOS), contrastada com a abordagem nativa direta (Ryu), é evidenciada na Fig. 4, que detalha o comportamento do alvo durante um surto intenso de inundação.

A análise das curvas na Fig. 4 consolida o impacto arquitetural de cada abordagem. A linha vermelha representa o controlador nativo (Ryu), que, por se comunicar puramente em OpenFlow, injeta a regra de bloqueio quase imediatamente após exceder o limiar de anomalia (em t=24s), suprimindo o tráfego malicioso em sua rampa inicial. Em contraste, a linha azul (ONOS via API) ilustra o custo prático do *overhead* sistêmico: o alerta gerado passa pela API *Northbound* e é processado pelo Núcleo Distribuído, gerando um atraso de transição. Essa breve janela permite que o ataque atinja um pico de 68.066 PPS antes do bloqueio efetivo no *hardware* em t=27s. Contudo, o gráfico comprova que ambos os paradigmas convergem para o mesmo estado de segurança, restaurando a interface ao fluxo nominal.

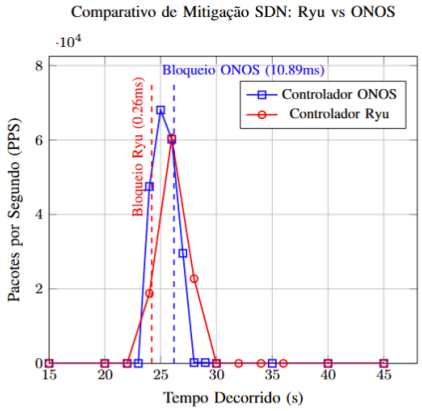

Figure 4. Comparativo do tempo de resposta e supressão de tráfego DDoS entre os controladores nativos (Ryu) e via API (ONOS). A inundação é neutralizada quase instantaneamente.

Como observado na Tabela II , mesmo com a carga massiva de 35,73 Mbps imposta pelo ataque SYN Flood, a interface retornou ao seu estado nominal (1 PPS) logo em seguida. Isso comprova empiricamente que, através da inserção de rotinas de tolerância a falhas na aplicação, a inteligência de software no controlador SDN é inteiramente capaz de sobrepor e compensar as latências e dessincronizações oriundas do hardware.

TABLE II. DESEMPENHO DA MITIGAÇÃO ATIVA SOB ATAQUE SYS FLOOD

Tempo (s)	PPS	Mbps	Estado do Sistema
10-23	1	0,00	Operação Nominal
24	47.510	27,96	Detecção de Anomalia
25	68.066	40,00	Pico de Inundação
26	60.198	35,73	**Instalação de Regra (ONOS)**
27	29.582	17,45	Queda de Tráfego (Supressão)
28	228	0,10	Mitigação Efetiva
30+	1	0,00	Fluxo Normalizado

E. *Sobrecarga da API REST e do Núcleo Distribuído*

A disparidade de latência observada entre o controlador nativo (~0.26 ms) e a abordagem via API *(~11 ms)* não reflete uma ineficiência de processamento bruto da rede, mas sim o custo inerente das abstrações de software do ONOS. Mesmo contornando o *Intent Framework*, a inserção de uma política via API REST na camada *Northbound* exige o processo de desserialização de cargas JSON e a validação de estado pelo Núcleo Distribuído (*Distributed Core*) antes de ser traduzida para *FlowMod* na interface *Southbound*. Em cenários de *botnets* altamente agressivas, esses milissegundos adicionais de

processamento transiente permitem que milhares de pacotes forjados atinjam o comutador antes que a mitigação faça efeito, reforçando a necessidade de defesas que tolerem a latência das chamadas REST.

F. Vulnerabilidade Topológica

Além do *overhead* sistêmico, as restrições físicas do ambiente expuseram um risco intrínseco de exaustão de infraestrutura. O sumário de topologia refletiu a operação centralizada de 14 *hosts* (e.g., originados das sub-redes 164.72.15.0/24 e 200.252.10.0/24) dependentes de um único comutador físico. Durante as avaliações de injeção Mirai-like com endereços MAC e IP forjados, o atacante gerou fluxos inéditos que invariavelmente causaram falha na correspondência de tabela (*Table Miss*).

Pelo comportamento reativo padrão do protocolo OpenFlow, cada cabeçalho de fluxo desconhecido acionou o encaminhamento de uma mensagem *Packet-In* para o controlador. Com 14 portas emulando ataques simultâneos em uma configuração *Top-of-Rack*, o evento culminou em uma Tempestade de Packet-In (*Packet-In Storm*). Esse fenômeno satura a CPU do ASIC e o canal de controle seguro antes mesmo da aplicação de mitigação iniciar o disparo de bloqueio (FlowMod com instrução Drop). Isso demonstra que, em *hardware* comercial, a eficácia do SDN como mitigador reativo de negação de serviço exige não apenas agilidade de código, mas políticas pré-instaladas de *Rate Limiting* no plano de dados para proteger o próprio cordão umbilical entre o *switch* e o controlador.

V. Conclusão

Este trabalho apresentou uma avaliação empírica de mecanismos de defesa SDN contra *botnets* IoT, utilizando infraestrutura de hardware comercial em um cenário hardware-in-the-loop. Os resultados confirmam que, embora a arquitetura SDN ofereça agilidade de resposta comparável a sistemas XDR modernos (com reações na ordem de milissegundos), sua aplicação prática na borda da rede enfrenta barreiras de confiabilidade significativas ligadas tanto à assincronia do plano de dados físico quanto ao *overhead* das abstrações de software.

A principal contribuição deste estudo foi demonstrar que o abismo entre simulações teóricas e a operação física vai além das limitações de memória do ASIC, como a persistência de regras *zumbis*. Constatamos que decisões de *design* na orquestração agravam o cenário: a busca por menor latência de reação via *bypass* do *Intent Framework* resulta na perda de reconciliação de estado nativa, enquanto a configuração topológica centralizada expõe o canal de controle a tempestades exaustivas de *Packet-In*.

Contudo, provamos empiricamente que tais limitações são contornáveis via software. A implementação de uma lógica robusta de verificação de estado (Malha Fechada) corrigiu integralmente as falhas de sincronização de identificadores e o acúmulo de regras retidas. Concluímos que a adoção de *switches bare-metal* comerciais na defesa autônoma de borda é plenamente viável e altamente eficaz, condicionada à utilização de controladores que incorporem inteligência para auditar ativamente o hardware e proteger seu próprio plano de controle contra a saturação.

VI. Agradecimentos

Os autores agradecem o apoio técnico e computacional do Laboratório LATITUDE, da Universidade de Brasília, a Secretaria Nacional de Assistência Social – SNAS/MDS, a Procuradoria Geral da Fazenda Nacional – PGFN, ao Projeto SISTER City – Sistemas Inteligentes Seguros e em Tempo Efetivo Real para Cidades Inteligentes ao projeto “Sistema de Controle e Unificação de Projetos para o Governo Distrito Federal – SISPRO-DF”, ao projeto “Pesquisa, Desenvolvimento e Aplicação de Protótipo do Ambiente de Simulação Didático para Veículos Ciberseguros, Autônomos e Conectados (SD-CyberAuto)”. Projeto “Pesquisa, Desenvolvimento e Aplicação – Metodologia para Apoio à Obtenção de Requisitos Éticos e de Privacidade” (Auxílio 514/2023) e a Fundação de Apoio do Distrito Federal - FAP/DF.

Referências Bibliográfica

[1] A. Antonakakis et al., “Understanding the Mirai Botnet," in Proc. 26th USENIX Security Symposium, Vancouver, BC, Canada, 2017, pp. 1093--1110.

[2] S. Scott-Hayward, G. O’Callaghan, and S. Sezer, “SDN Security: A Survey," IEEE Communications Surveys \& Tutorials, vol. 18, no. 1, pp. 623--645, Firstquarter 2016.

[3] P. Berde et al., “ONOS: The Open Network Operating System," IEEE Communications Magazine, vol. 52, no. 11, pp. 107--117, 2014.

[4] A. Tootoonchian et al., “On Controller Performance in Software-Defined Networks," in Proc. 2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE), San Jose, CA, USA, 2012.

[5] J. Ben-Othman et al., “Performances of OpenFlow-Based Software-Defined Networks: An overview," Journal of Networks, vol. 10, no. 10, pp. 563--574, 2015.

[6] S. O. A. Mohammed, “Security Analysis of ONOS Software-Defined Network Platform," Ontario Tech University, Technical Report, 2017.

[7] A. S. Muqaddas et al., “Inter-Controller Traffic to Support Consistency in ONOS Clusters," IEEE Transactions on Network and Service Management, vol. 14, no. 4, pp. 1018--1031, 2017.

[8] R. Braga, E. Mota, and A. Passito, “Lightweight DDoS Flooding Attack Detection Using NOX/OpenFlow," in Proc. IEEE Local Computer Networks (LCN), Denver, CO, USA, 2010, pp. 408--415.

[9] K. Giotis et al., “Combining OpenFlow and sFlow for an effective and scalable Anomaly Detection and Mitigation mechanism on SDN Environments," Computer Networks, vol. 62, pp. 122--136, 2014.

[10] S. Wijeratne et al., “Scalable High Performance SDN Switch Architecture on FPGA for Core Networks," arXiv preprint arXiv:1910.13683, 2019.

[11] I. M. Meidan et al., “N-BaIoT: Network-based Detection of IoT Botnet Attacks Using Deep Autoencoders," IEEE Pervasive Computing, vol. 17, no. 3, pp. 12--22, 2018.

[12] K. Shinan et al., “Machine Learning-Based Botnet Detection in Software-Defined Network: A Systematic Review," Computers & Security, vol. 122, p. 102871, 2022.

[13] Dell Networking, “S3048-ON Specification Sheet," Dell Inc., Round Rock, TX, USA, 2016.

[14] M. F. Romani et al., “Mitigação de ataques ddos em redes iot com aprendizado federado," Revista Ibérica de Sistemas e Tecnologias de Informação, no. E77, pp. 160--172, 2025.

[15] K. H. R. Mbongo et al., “Conv1d-gru-self attention: An efficient deep learning framework for detecting intrusions in wireless sensor networks," Future Internet, vol. 17, no. 7, pp. 1--20, 2025.