Análise de Desempenho e Eficácia da Sonda Zeek: Um Estudo Comparativo de Perfis de Execução sob Restrições de Recursos

Rafael B Torres[1] **, Rodrigo Brandão Mansilha**[1] **, Diego Kreutz**[1]

[1]AI Horizon Labs – Programa de Pós-Graduação em Engenharia de Software (PPGES)
Universidade Federal do Pampa (UNIPAMPA)

{rafaelbarboza.aluno, rodrigomansilha, diegokreutz}@unipampa.edu.br

Abstract. *This work evaluates the performance and efficacy of Zeek as a security probe within the GT-IoTEdu project, considering its superior potential for behavioral analysis compared to signature-based NIDS. The experiment involves five network attacks, two Zeek execution modes (Standard Mode and Intel Framework Mode), and four hardware environments simulated via Docker, ranging from limited IoT devices to robust servers. The results demonstrate total detection efficacy and viability on devices with 1 GB of RAM (Raspberry Pi 4), being unfeasible only on low-memory routers (<256 MB), where the Intel Framework impacted latency. It is noteworthy that the analysis was based on resource emulation, not verifying the native performance of the ARM architecture, a step reserved for future work.*

Resumo. *Este trabalho avalia o desempenho e a eficácia do Zeek como sonda de segurança no projeto GT-IoTEdu, considerando seu potencial superior de análise comportamental em comparação com NIDS baseados em assinatura. O experimento envolve cinco ataques de rede, dois modos de execução do Zeek (Modo Padrão e Modo Intel Framework) e quatro ambientes de hardware simulados via Docker, variando de dispositivos IoT limitados a servidores robustos. Os resultados demonstram eficácia total de detecção e viabilidade em dispositivos com 1 GB de RAM (Raspberry Pi 4), sendo inviável apenas em roteadores de baixa memória (<256 MB), onde o Intel Framework impactou a latência. Ressalta-se que a análise se baseou em emulação de recursos, não verificando o desempenho nativo da arquitetura ARM, etapa reservada para trabalhos futuros.*

1. Introdução

A proliferação de dispositivos IoT tem ampliado de forma significativa a superfície de ataque e introduzido novos vetores de ameaça. Um exemplo marcante ocorreu em 2016, quando a botnet Mirai explorou dispositivos vulneráveis para lançar um dos maiores ataques de DDoS já registrados [Antonakakis et al. 2017]. Situações como essa evidenciam a necessidade de ferramentas capazes de monitorar e analisar tráfego em tempo real, como Suricata, Snort e Zeek, que permitem identificar comportamentos anômalos e detectar atividades maliciosas antes que comprometam a rede.

Nas universidades brasileiras, o aumento de dispositivos IoT conectados transformou laboratórios, pesquisas e infraestrutura em ambientes ciberfísicos complexos, ampliando a demanda por mecanismos robustos de monitoramento e controle. O cenário

institucional é marcado por grande heterogeneidade: enquanto UFRGS e Unicamp adotam processos formais, embora lentos, de homologação de dispositivos, instituições como UNIPAMPA e UFAM não possuem políticas estruturadas para gestão de IoT. A falta de padronização gera práticas improvisadas, inconsistências operacionais e maior exposição a vulnerabilidades, sobretudo em campi que reúnem centenas de dispositivos sem coordenação centralizada. Nesse contexto, o IoTEdu[1] propõe um modelo unificado de registro, controle de acesso e monitoramento contínuo, adaptado às necessidades das instituições acadêmicas brasileiras. Entre seus objetivos está a implementação de sondas de segurança capazes de operar em diferentes pontos da rede, utilizando ferramentas como Suricata, Snort e Zeek, desde roteadores até dispositivos IoT com recursos restritos.

Neste trabalho, realiza-se uma avaliação de desempenho da ferramenta Zeek para investigar sua viabilidade como um dos componentes centrais de monitoramento no IoTEdu, examinando a relação entre custo computacional e capacidade de detecção. O estudo considera três dimensões: (i) a eficácia do Zeek frente a cinco vetores de ataque distintos; (ii) o impacto sobre CPU, memória e latência quando executado em quatro perfis de hardware simulados via Docker, variando de dispositivos de borda a servidores mais robustos; e (iii) a sobrecarga introduzida pelo Modo Intel Framework em comparação ao modo desabilitado, permitindo quantificar o custo associado à correlação de inteligência de ameaças em tempo real.

2. Trabalhos Relacionados

A literatura sobre monitoramento de redes e detecção de intrusão frequentemente compara três classes principais de ferramentas: sistemas baseados em assinatura, como o Snort; motores multithread com processamento otimizado, como a Suricata; e plataformas orientadas à análise comportamental e telemetria, como o Zeek [Waleed et al. 2022, san 2024]. Estudos específicos sobre o Zeek destacam que sua arquitetura orientada a eventos e sua produção de logs transacionais (por exemplo, `conn.log`, `http.log`, `dns.log`) o tornam especialmente adequado para investigação pós-incidente e monitoramento contínuo, embora seu custo computacional varie de acordo com o volume de tráfego e a configuração de workers e threads [zee 2025, project (GitHub) 2024].

No contexto de redes IoT, pesquisas recentes investigam a viabilidade de empregar sondas como o Zeek em ambientes de recursos limitados. Trabalhos como os de [Huda et al. 2025, Farag et al. 2025, report) 2025] demonstram que o Zeek é capaz de detectar exfiltração por DNS e outros comportamentos anômalos nesses cenários, mas ressaltam que sua execução em dispositivos com memória muito reduzida (como roteadores OpenWRT com 128–256 MB) apresenta limitações relevantes. Em contrapartida, plataformas como Raspberry Pi 4 e máquinas virtuais leves mostram-se adequadas para atuação como sondas de borda, desde que se considere o equilíbrio entre latência, taxa de eventos e funcionalidades adicionais, incluindo o uso do *Intel Framework*.

A literatura também discute o impacto da integração com fontes de inteligência de ameaças (IoCs). A documentação oficial e relatos de implantação apontam que o *Intelligence Framework* do Zeek facilita a correlação entre indicadores e eventos observados, mediante um mecanismo projetado para otimizar memória e desempenho. Ainda assim, notas técnicas e estudos de caso relatam aumentos mensuráveis de uso de CPU e memória

[1] `https://gt-iotedu.github.io`

quando grandes conjuntos de IoCs são carregados e processados em tempo de execução [zee 2025, Grashöfer 2016].

Outros trabalhos avaliam o desempenho de sistemas de detecção de intrusão, como Suricata, Snort e Zeek, porém o fazem predominantemente sob a perspectiva de eficácia de detecção, analisando métricas como taxa de acerto, cobertura e falso-positivos [Ghazi et al. 2024, Boukebous et al. 2023, Park and Ahn 2017, Murphy 2019]. No entanto, poucos estudos investigam esses IDS sob a ótica do consumo de recursos computacionais, como utilização de CPU, memória e impacto no fluxo de rede, especialmente em cenários com restrições de hardware. Essa lacuna é particularmente relevante para ambientes IoT e gateways de borda, nos quais a disponibilidade de recursos é limitada e o custo operacional do IDS torna-se um fator crítico para sua adoção prática.

Em síntese, embora a literatura recente explore o uso do Zeek em IoT e o overhead geral de IoCs, observa-se uma lacuna na quantificação isolada do custo computacional do Intelligence Framework sob restrições granulares de recursos em ambientes conteinerizados. A maioria das abordagens existentes foca na comparação generalista entre diferentes NIDS ou em implementações em hardware físico específico, sem detalhar os limiares de saturação ao correlacionar ameaças em tempo real. Este trabalho motiva-se, portanto, pela necessidade de preencher essa lacuna, avaliando sistematicamente o trade-off entre capacidade de detecção e consumo de recursos para validar a aplicabilidade da sonda no cenário heterogêneo do projeto GT-IoTEdu.

3. Metodologia

Esta seção descreve a metodologia utilizada para conduzir um *benchmark* de três fatores (Recursos, Ataques e Perfil de Execução) aplicado à sonda Zeek. O planejamento priorizou a reprodutibilidade dos experimentos, permitindo que diferentes pesquisadores repliquem os resultados com mínima variação ambiental.

3.1. Ambiente

A topologia de testes foi construída com virtualização via Docker, garantindo isolamento, controle preciso de recursos e repetibilidade. A arquitetura utiliza uma rede virtual do tipo bridge denominada rede-alvo, que interliga todos os contêineres envolvidos no experimento, incluindo o atacante, o servidor alvo e a sonda Zeek. Nessa configuração, o Zeek opera em modo passivo e promíscuo, recebendo uma cópia integral do tráfego entre atacante e alvo, sem a necessidade de atuar como gateway ou participar do roteamento.

3.2. Parâmetros

O desempenho da sonda foi avaliado por meio de um desenho experimental composto por três fatores. O experimento combina quatro níveis de capacidade de hardware virtualizado, dois perfis de execução do Zeek e cinco ataques distintos, totalizando 40 cenários de teste.

Capacidade de hardware (emulado). Para medir a sobrecarga introduzida pelo Zeek, foram emuladas quatro configurações de implantação por meio de limites de CPU e memória aplicados no arquivo `docker-compose.yml` com os parâmetros `cpu_limit` e `mem_limit`. Todas as execuções ocorreram no mesmo host físico (40 GB RAM, Intel Core i7), garantindo que apenas as restrições do contêiner variassem entre os cenários.

As configurações foram: ($C1$) 1 núcleo de CPU e 1 GB RAM, aproximando dispositivos como Raspberry Pi 4; ($C2$) 2 núcleos e 2 GB RAM, simulando máquinas virtuais leves; ($C3$) 4 núcleos e 4 GB RAM, representando servidores de borda; ($C4$) 8 núcleos e 40 GB RAM, atuando como referência de base.

Modos de operação. O Zeek foi configurado para gerar todos os logs transacionais e executar todos os analisadores de protocolo, variando apenas o estado do Intelligence Framework. Dois perfis foram avaliados: ($O1$) desabilitado, com operação padrão; e ($O2$) habilitado, carregando um conjunto de indicadores de comprometimento para correlação ativa.

Ataques. A A Tabela 1 apresenta os cinco ataques utilizados no experimento, executados em contêineres isolados para garantir reprodutibilidade. Os ataques cobrem diferentes classes de ameaças típicas de redes IoT e ambientes campus, e incluem o contêiner, as ferramentas e uma breve descrição da ação realizada.

Tabela 1. Descrição detalhada dos Contêineres de Ataque

Contêiner	Ataque	Ferramentas	Descrição da Ação Sucinta
`dos-http`	($A1$) DoS Attack	`ab`, `curl`, `slowloris`	Ataque DoS multi-vetor (+12k reqs) combinando GET Flood, POST Flood, Slowloris (conexões lentas) e Header Flood (headers de 2KB).
`brute-force-ssh`	($A2$) Brute Force SSH	`hydra`	Usa **`hydra`** para 100 tentativas de login SSH ('root') com senhas aleatórias.
`icmp-flood`	($A3$) ICMP Flood	`hping3`	Executa ICMP flood por 10s com **`hping3`**. Usa pacotes de 1200 bytes na velocidade máxima (`-flood`).
`dns-tunneling`	($A4$) DNS Tunneling	`dig` (dnsutils)	Envia 200 consultas DNS (`dig`) com subdomínios longos (12-62 chars) e alta entropia (hex) para simular exfiltração de dados.
`sql-injection`	($A5$) SQL Injection	`sqlmap`	Executa **`sqlmap`** contra a URL alvo em modo *batch* e *level 3* para explorar falhas de SQLi.

3.3. Métricas e Instrumentos de Medição

As métricas de avaliação foram organizadas em duas categorias: eficácia, relativa à detecção de ataques, e eficiência, referente ao uso de recursos. Para a eficácia, avaliou-se: (1) a capacidade de detecção (resultado binário Sim/Não); e (2) a riqueza de contexto fornecida pelos logs, comparando a profundidade forense do Modo *Inteligence Framework* desabilitado (logs transacionais) com a do modo habilitado (logs de correlação de ameaças). Para a eficiência, mensurou-se: (3) o pico de consumo de recursos (CPU em % por núcleo e RAM em MB); e (4) a latência de alerta, definida como o tempo decorrido entre o início do ataque e o primeiro registro relevante nos logs de notificação.

O monitoramento de recursos (CPU e RAM) foi realizado utilizando o comando nativo `docker stats`. Scripts de aferição de tempo (baseados em *timestamps* de início de ataque e de geração de log) foram usados para medir a latência dos alertas.

4. Resultados

O Zeek manteve desempenho consistente em todos os cenários avaliados. Independentemente da capacidade de hardware emulada, variando de dispositivos de baixo custo ($C1$) a servidores robustos ($C4$), e independentemente do uso do Intelligence Framework (habilitado ou desabilitado), a sonda detectou corretamente todos os cinco ataques testados: DoS, Brute Force SSH, ICMP Flood, DNS Tunneling e SQL Injection. Esses resultados

indicam que a detecção do Zeek não é sensível às restrições de CPU e memória impostas nos experimentos e que a ativação do Intelligence Framework não compromete sua eficácia. Isso reforça sua adequação como sonda de segurança em ambientes com ampla diversidade de recursos, como campi acadêmicos e redes IoT heterogêneas.

A Tabela 2 mostra que ataques de maior volume, como DoS e ICMP Flood, elevam significativamente o consumo de CPU, especialmente em cenários com poucos recursos, enquanto vetores menos intensivos apresentam impacto modesto. A ativação do Intelligence Framework aumenta o uso de memória e pode elevar a latência, embora o comportamento da CPU permaneça proporcional ao tipo de ataque e à capacidade disponível. No geral, o Zeek mantém operação estável mesmo em dispositivos de baixo custo, mas com maior custo de desempenho diante de tráfego volumétrico e correlação ativa de inteligência.

Tabela 2. Impactos dos parâmetros no uso de recursos (CPU, RAM, Latência).

Vetor de Ataque	Cenário	Inteligence Framework Desabilitado			Inteligence Framework Habilitado		
		CPU [Pico]	RAM [Pico]	Latência (s)	CPU [Pico]	RAM [Pico]	Latência (s)
DoS Attack	$C1$ Baixo	102.48%	139.9 MB	7s	101.14%	160.1 MB	13s
	$C2$ Médio	109.96%	137.3 MB	7s	108.44%	159.9 MB	8s
	$C3$ Alto	110.15%	137.2 MB	6s	109.49%	157.7 MB	8s
	$C4$ Base	109.55%	138.2 MB	8s	112.02%	157.1 MB	8s
SQL Injection	$C1$ Baixo	13.90%	130.8 MB	2s	12.41%	159.5 MB	4s
	$C2$ Médio	12.96%	127.9 MB	2s	13.72%	148.0 MB	2s
	$C3$ Alto	12.57%	128.0 MB	2s	13.35%	147.9 MB	2s
	$C4$ Base	12.91%	128.0 MB	1s	12.65%	147.7 MB	1s
DNS Tunneling	$C1$ Baixo	3.31%	143.8 MB	2s	2.79%	159.3 MB	4s
	$C2$ Médio	2.90%	127.9 MB	0s	1.94%	148.1 MB	1s
	$C3$ Alto	1.86%	128.0 MB	0s	2.12%	147.9 MB	0s
	$C4$ Base	2.28%	127.9 MB	0s	2.10%	148.0 MB	0s
ICMP Flood	$C1$ Baixo	13.86%	130.8 MB	3s	14.28%	159.3 MB	7s
	$C2$ Médio	14.76%	127.9 MB	0s	12.73%	148.2 MB	1s
	$C3$ Alto	15.02%	128.1 MB	0s	12.97%	149.9 MB	0s
	$C4$ Base	16.61%	127.9 MB	0s	13.72%	147.7 MB	0s
Brute Force	$C1$ Baixo	3.31%	130.8 MB	1s	1.11%	159.4 MB	3s
	$C2$ Médio	2.22%	128.0 MB	1s	2.28%	148.2 MB	1s
	$C3$ Alto	1.73%	128.0 MB	0s	2.14%	148.1 MB	0s
	$C4$ Base	1.88%	128.2 MB	0s	2.30%	147.8 MB	0s

Impacto do Intel Framework no uso de memória e unidade de processamento: Em todas as quatro capacidades e nos cinco tipos de ataque, o Modo Intel Framework consumiu consistentemente cerca de 20-30 MB a mais de RAM do que quando desabilitado. Nos ataques de baixa intensidade, onde o $C4$ (Base de comparação) estabilizou em 128 MB para o Modo Intel Framework desabilitado, estabilizou em 148 MB. Isso representa um *overhead* de memória estático de aproximadamente 15-20% apenas para manter o framework de inteligência carregado e pronto para correlação. Em contrapartida, o impacto na CPU para esses mesmos ataques de baixa intensidade foi desprezível, com ambos os modos apresentando picos de consumo muito similares e baixos (entre 1% e 16%).

Impacto da Restrição de Recursos na Latência: O verdadeiro impacto do Intel Framework tornou-se evidente sob estresse e restrição de recursos. No **Cenário 1 (Baixo)**, que simula um dispositivo de borda (1 núcleo, 1GB RAM), a latência para geração de alertas foi drasticamente afetada. Durante o ataque de DoS (Tabela 2), a latência do "Modo desabilitado" foi de 7 segundos, enquanto o "Modo habilitado Framework" quase dobrou, saltando para 13 segundos. O mesmo padrão se repetiu no ICMP Flood, com 3s (desabilitado) vs. 7s (habilitado), e nos ataques de SQLi e DNS, onde a latência também dobrou de 2s para 4s. Isso demonstra que, sob contenção de CPU, a carga adicional do Intel Framework, mesmo que pequena, compete por ciclos de processamento e impacta diretamente a velocidade de resposta da sonda.

Viabilidade e Pontos de Saturação: Em todos as quatro capacidades, o pico de CPU ultrapassou 100%, indicando que a sonda Zeek saturou o núcleo de CPU disponível (ou mais de um, nos cenários maiores) para processar o *flood* de tráfego. Curiosamente, o consumo de CPU não aumentou significativamente nos cenários com mais núcleos, e a latência se estabilizou a partir do Cenário 2 (Médio), com 7-8s. Isso sugere que, para esta carga de trabalho específica, 1 núcleo (Cenário 1) é um gargalo claro que degrada a latência, mas 2 núcleos (Cenário 2) já são suficientes para lidar com o processamento sem degradação adicional.

Viabilidade em Dispositivos de Borda (OpenWRT e Raspberry Pi): Os resultados permitem avaliar a aplicabilidade da sonda em testbeds reais do GT-IoTEdu. A instalação direta em roteadores com OpenWRT mostra-se inviável, já que o consumo mínimo de RAM da sonda, com pico de aproximadamente 130MB no Cenário 1, excede a memória total disponível na maioria desses dispositivos, normalmente entre 128MB e 256MB. Em contraste, o Cenário 1 (Baixo) confirma o Raspberry Pi 4, com 1GB de RAM, como uma alternativa viável, pois o pico de uso próximo de 160MB é plenamente suportado. Entretanto, a saturação de CPU em torno de 102 por cento e a latência elevada entre 7 e 13 segundos durante ataques DoS indicam que, embora operacional, a sonda trabalha próxima ao limite. A ativação do Modo Intel Framework nesse hardware restrito aumenta significativamente o tempo de resposta do alerta, evidenciando um claro trade-off entre capacidade de inteligência e desempenho em tempo real.

5. Conclusão

Este trabalho avaliou o desempenho e a eficácia do Zeek como sonda de segurança em diferentes capacidades de hardware no contexto do GT-IoTEdu. O benchmark combinou dois perfis de execução e quatro níveis de recursos contra cinco ataques, obtendo detecção completa em todos os cenários. Os resultados mostram que o Zeek é inviável para roteadores OpenWRT com menos de 256 MB de RAM, mas plenamente utilizável em dispositivos com 1 GB de RAM, como o Raspberry Pi 4, nos quais o impacto do Intelligence Framework permanece moderado e operacionalmente seguro quando há ao menos dois núcleos de CPU disponíveis.

Apesar da robustez dos resultados, a avaliação apresenta limitações relevantes. A emulação de recursos ocorreu sobre arquitetura x86, não refletindo fielmente o comportamento de processadores ARM amplamente utilizados em IoT, e o tráfego sintético em ambiente controlado não captura toda a variabilidade e ruído de redes reais. Como continuidade, pretende-se validar os achados em hardware físico ARM, desenvolver scripts

de detecção específicos para IoT, otimizar o Intelligence Framework via filtragem seletiva de IoCs e integrar os logs ao IoTEdu e a sistemas SIEM para possibilitar visualização, correlação e análise em tempo real.

Agradecimentos

Esta pesquisa recebeu apoio parcial da RNP[2], por meio do GT IoTEdu[3], e da CAPES[4], sob o Código de Financiamento 001.

Referências

(2024). Evaluating the efficacy of network forensic tools: Comparative analysis of snort, suricata, and zeek.

(2025). Intelligence framework — book of zeek. `https://docs.zeek.org/en/master/frameworks/intel.html`. Documentação oficial do Zeek — seção Intelligence Framework.

Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cestio, L., Dickerson, R. D., Paxson, V., Rioja, J. R., Saha, S., Svoboda, D., Ullrich, J. D., Weiss, M., and Durumeric, Z. (2017). Understanding the Mirai botnet. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1093–1110. USENIX Association.

Boukebous, A. A. E., Fettache, M. I., Bendiab, G., and Shiaeles, S. (2023). A comparative analysis of snort 3 and suricata. In *2023 IEEE IAS Global Conference on Emerging Technologies (GlobConET)*, pages 1–6. IEEE.

Farag, W. et al. (2025). Development and evaluation of a novel iot testbed for ...

Ghazi, D. S., Hamid, H. S., Zaiter, M. J., and Behadili, A. S. G. (2024). Performance and efficacy of snort versus suricata in intrusion detection: A benchmark analysis. In *AIP Conference Proceedings*, volume 3232, page 020024. AIP Publishing LLC.

Grashöfer, J. (2016). The intelligence framework update. `https://zeek.org/2016/12/the-intelligence-framework-update/`.

Huda, S., Musthafa, M. B., and Nogami, Y. (2025). A performance evaluation of zeek-based intrusion detection in agricultural iot security.

Murphy, B. R. (2019). *Comparing the performance of intrusion detection systems: Snort and Suricata*. PhD thesis, Colorado Technical University.

Park, W. and Ahn, S. (2017). Performance comparison and detection analysis in snort and suricata environment. *Wireless Personal Communications*, 94(2):241–252.

project (GitHub), Z. (2024). Zeek benchmarkers.

report), M. D. H. D. (2025). Evaluating zeek and suricata for intrusion detection — thesis/report.

Waleed, A., Jamali, A. F., and Masood, A. (2022). Which open-source ids? snort, suricata or zeek. *Computer Networks*, 213:109116.

[2]`https://www.rnp.br`
[3]`https://gt-iotedu.github.io`
[4]`https://www.gov.br/capes/pt-br`