Filtragem de Pacotes na Borda da Rede: uma Análise Comparativa com Foco no Consumo de Energia

Arthur Eugenio Silverio[1], Hélio Crestana Guardia[1]

[1] Departamento de Computação – Universidade Federal de São Carlos (UFSCar)

Caixa Postal 676 – 13.565-905 – São Carlos, SP – Brazil

arthur.silverio@estudante.ufscar.br, helio.guardia@ufscar.br

Abstract. *Edge Security is concerned with protecting not only the devices connected at the borders of the network, but also the core of the network from the traffic generated at these devices. Threat detection systems analyze the behavior of edge computing devices for potential threats and apply policies as early as possible, before they scale the infrastructure. Many aspects make edge security in edge particularly sensitive due to the low computational capacity of devices and the need to aggressively minimize energy consumption. In this sense, this paper presents the results of a comparative analysis of packet filtering using different operating system hooks in a Linux system, ranging from as close to the interface device to using userspace tools. Besides evaluating the CPU load, it was also measured energy consumption in each case. Analysis shows the importance of packet processing always taking place at the lowest layers of the operating system.*

Resumo. *O conceito de segurança de borda procura não apenas proteger os dispositivos na borda na rede, mas também o núcleo da rede do tráfego gerado nesses dispositivos. Os sistemas de detecção de intrusão analisam o comportamento dos dispositivos de borda em busca de ameaças e aplicam políticas o mais cedo possível, antes que escalem a infraestrutura de nuvem. Muitos aspectos tornam a segurança de borda sensível, particularmente, a baixa capacidade computacional dos dispositivos e a necessidade de minimizar agressivamente o consumo de energia. Nesse sentido, este artigo apresenta uma análise comparativa da filtragem de pacotes utilizando diferentes ganchos de sistema operacional em um sistema Linux, desde os ganchos mais próximos da interface até o uso de ferramentas do espaço do usuário. Além de avaliar a carga da CPU, também foi medido o consumo de energia associado. A análise mostra a importância de o processamento de pacotes ocorrer sempre nas camadas mais baixas do sistema operacional.*

1. Introdução

Ao longo do tempo a Internet tem cada vez mais se mostrado a ferramenta capaz de tornar o conceito de "inteligência interconectada" uma realidade, utilizando estratégias que interligam dispositivos heterogêneos entre si de modo escalável do núcleo central da rede aos dispositivos mais periféricos [MITRA et al. 2021]. Como se vê na já consolidada tecnologia de telefonia celular 5G, paradigmas atuais de comunicação e processamento estendem o uso de *data centers*, ou da arquitetura física que abriga os sistemas computacionais de uma corporação ou aplicação, usando técnicas de virtualização em nuvem e arquitetura de microsserviços [PORAMBAGE et al. 2021].

Este modelo de processamento é pertinente em diferentes domínios em que números variados de dispositivos com diferentes capacidades de processamento são interligados em rede, gerando dados e consumindo os resultados de seus processamentos. Nesses cenários existe uma tendência de que os dados gerados pelos

dispositivos sejam processados nas bordas, para que a comunicação com o interior da rede seja evitada na hora de consumir as informações extraídas de seus processamentos. Nesse contexto é possível que sensores existam em grandes quantidades nas bordas da rede, por exemplo, e que tenham capacidade de processamento reduzida e restrições energéticas. Também pode ocorrer de os dados serem gerados com elevada frequência e que o tempo hábil para diagnóstico de eventuais falhas nos equipamentos, ou outras condições avaliadas em tempo real, seja reduzido, de forma que enviar os dados para processamento centralizado pode gerar elevada carga na rede de interconexão.

Como a confiabilidade é fundamental nesses cenários, a heterogeneidade dos dispositivos revela novos problemas de segurança, além daqueles tradicionais, que se tornam mais evidentes e mais graves. A análise e o processamento de pacotes são essenciais para que o encaminhamento, a detecção e mitigação de ataques e de tentativas de acesso indevidas ocorram da forma mais rápida e eficiente possível. O conceito de *edge security* aplica a segurança em dispositivos de borda, a fim de proteger tanto os dispositivos que estão conectados no perímetro da rede quanto para impedir que ataques provenientes dos dispositivos de borda escalem a rede central. Os problemas da segurança de borda devem-se no geral à baixa capacidade computacional dos dispositivos *low-end*, ao desconhecimento da natureza dos ataques, à heterogeneidade dos sistemas operacionais e protocolos e à alta granularidade do controle de acesso de modelos tradicionais [XIAO et al. 2019]. Os sistemas de detecção inteligente de ameaças analisam o comportamento dos dispositivos de computação de borda contra possíveis ameaças e aplicam políticas o mais cedo possível baseando-se nessas análises.

Se nos *desktops* e dispositivos móveis os recursos de *hardware* estão ficando cada vez mais poderosos, nos dispositivos de *low-end*, que podem ser comuns nas bordas da rede, existe sempre a necessidade constante de minimizar o custo de todos os componentes e de conservar agressivamente energia, memória, CPU e largura de banda. A economia energética se tornou crucial para todas as indústrias, principalmente no universo da indústria da tecnologia [JIANG et al. 2021].

Tendo em vista o uso de um sistema *Linux* como elemento intermediário para a interligação de dispositivos na borda de uma rede, esse artigo foca na quantificação do consumo energético de técnicas de filtragem de pacotes. Análises realizadas neste trabalho mostram as vantagens de o processamento de pacotes ocorrer nas camadas mais baixas do sistema operacional, de forma que o pacote atravesse a menor quantidade de camadas possíveis dentro da pilha de protocolos. Para avaliar estas questões, este trabalho apresenta um estudo comparativo sobre o desempenho e o impacto energético na filtragem de pacotes de rede, considerando ferramentas e estratégias com atuação em diferentes camadas do sistema operacional. Nas avaliações realizadas foram comparadas várias técnicas de filtragem para cargas equivalentes. Para análise de pacotes foram comparados técnicas de filtragem em espaço de usuário e filtros com *BPF* e *eBPF*; já para o processamento de pacotes foram comparados mecanismos baseados em *netfilter*: *iptables* e *nftables*, além de *traffic control (tc)* e *XDP*. Usando um computador *Raspberry PI*, o estudo avalia o impacto no uso de recursos computacionais (*CPU*) e gastos energéticos associados.

O restante do artigo está organizado da seguinte maneira: na seção 2 são citados

artigos e trabalhos relacionados que serviram de motivação para este trabalho; na seção 3 é apresentado um breve histórico com detalhamento de mecanismos de filtragem de pacotes no sistema Linux; na seção 4 é apresentado o estudo de caso; na seção 5 são descritos os cenários analisados nos testes; na seção 6 são apresentados os resultados, seguindo-se a conclusão na última seção.

2. Trabalhos Relacionados

Em [VIEIRA et. al 2021] os autores apresentam as ferramentas para filtragem de alto desempenho com foco no *eBPF* em comparação com o antigo BPF do Unix BSD. No artigo [MELKOV et al. 2020] os autores comparam o desempenho da filtragem de pacotes de *iptables* e *nftables* em diferentes cadeias de regras (*chains*) e tabelas. Em [HØILAND-JØRGENSEN et al. 2018], os autores apresentaram o design *XDP* e forneceram uma avaliação de desempenho comparando os modelos *XDP, DPDK, raw* e *conntrack* na pilha de rede. Em [SCHOLZ et al. 2018], os autores produziram uma detalhada análise de desempenho da filtragem de pacotes usando *eXpress Data Path (XDP)* e *eBPF*, comparando diferentes cenários e os desempenhos dos programas executando com *JIT* (*Just-in-Time Compilation*) e usando o interpretador no kernel (Sem *JIT*).

Complementarmente aos trabalhos anteriores citados, este artigo procura realizar um estudo comparativo de técnicas de processamento de pacotes, introduzindo a medição dos consumos energéticos associados aos processamentos. Em cenários com dispositivos de baixa capacidade de processamento e com restrições energéticas, a seleção de mecanismos de filtragem com menor consumo pode ser essencial.

3. Mecanismos de filtragem de pacotes

A Figura 1 ilustra estratégias de filtragem e seus posicionamentos na pilha de protocolos do sistema Linux.

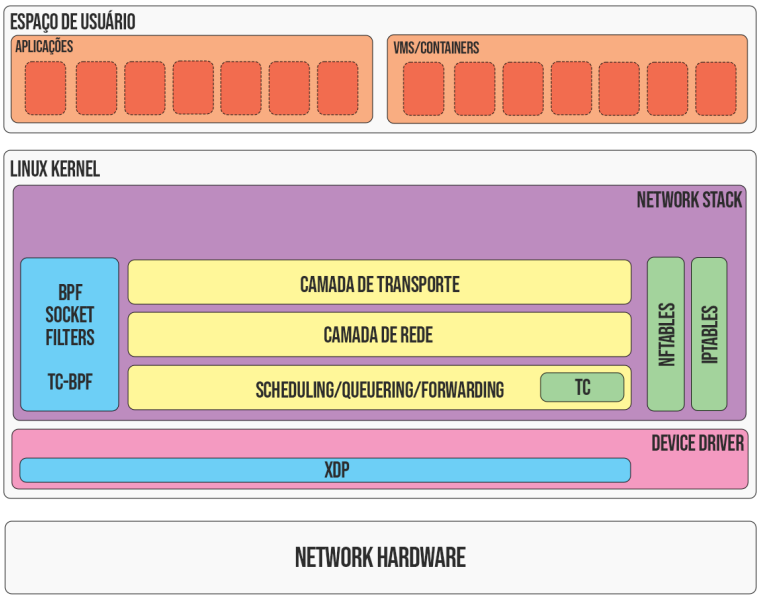

Figura 1. Camadas da pilha de protocolos e ferramentas de filtragem

No funcionamento da pilha de protocolos de rede de um sistema operacional como o Linux, há diferentes mecanismos e estratégias para filtragem de pacotes, que podem ocorrer desde o recebimento de um quadro pelo controlador de dispositivo, até o momento em que as informações do pacote já estão disponíveis em espaço de usuário para tratamento por aplicações específicas. Do ponto de vista apenas de descarte de pacotes, essas estratégias podem ser consideradas equivalentes.

3.1. BPF

A era da filtragem de pacotes de alto desempenho começou com os pesquisadores Steven McCanne e Van Jacobson da University of California *Berkeley* [Mccane and Jacobson 1993], que apresentaram um novo modelo para filtragem de pacotes no nível do usuário. Neste trabalho, os autores argumentam que os *sniffers* de rede que são executados como processos no nível do usuário copiam os pacotes fazendo com que atravessem do espaço do *kernel* para o espaço de usuário. Esse modelo de cópia cria uma sobrecarga que pode ser diminuída pelo filtro baseado em uma máquina de registradores que descarta o pacote o mais cedo possível, evitando que dados sejam copiados desnecessariamente, ou que comparações redundantes sejam realizadas. Comparando o desempenho do modelo *BPF* proposto e do antigo Sun *NIT*, os autores explicam porque os *CFGs* (*Control Flow Graphs*) usados pelo BPF eram mais eficientes do que as árvores usadas por Sun NIT, que faziam comparações redundantes antes de descartar um pacote. Esse modelo já virtualiza suas funções de rede para dentro do *kernel*, mesmo que de forma local.

eBPF é uma tecnologia baseada no *kernel Linux,* que é uma evolução do *BPF* dos sistemas *BSD*, que permite a execução de código arbitrário dentro de uma máquina virtual isolada dentro do *kernel* do sistema operacional sem a necessidade de se modificar o código fonte do *kernel*. Originalmente, *BPF* criado no sistema *unix BSD* oferece suporte à filtragem de pacotes de maneira que um processo sendo executado em espaço de usuário forneça um programa de filtro que especifique quais pacotes deseja receber. *eBPF* foi projetado para que o *BPF* recebesse algumas extensões e fosse usado com um compilador em tempo de execução (*JIT*) com mapeamento direto um para um dos registradores do processador.

3.2. Netfilter

Netfilter é uma estrutura fornecida pelo *kernel Linux* que permite acesso às funções de *firewall*, *NAT* (*network address translation*) e *logs*, e que permite manipular os pacotes que atravessam a pilha de protocolos no *kernel*. O *Netfilter* utiliza ganchos presentes dentro da própria pilha de rede do *kernel* que permitem que módulos específicos registrem funções de retorno de chamada com a pilha de rede do *kernel*. As regras são tratadas como rotinas de interrupções que são chamadas para cada pacote que atravessa o respectivo gancho (*hook*) dentro da pilha de rede. As *tables* são: *pre-routing*, *forward*, *input*, *output*, *post-routing*. As *chains* são: *filter*, *route* e *nat*.

Historicamente as regras de segurança no filtro de pacotes em sistemas *Linux* são configuradas por meio do programa *iptables*, que é um software de *firewall* que permite definir conjuntos de regras dentro do mecanismo de filtro de pacotes (*netfilter*). Cada regra dentro de uma tabela consiste em um número de classificadores e uma ação

conectada. Mas atualmente, a configuração de regras de *firewall* em sistemas *Linux* tem sido gradativamente direcionada ao programa sucessor *nftables*. O *nftables* tem uma arquitetura mais simplificada e um novo modo de aplicar as regras dentro dos ganchos do *kernel*, além de ser *protocol-agnostic*, escalável e possuir uma maior integração com outros componentes do *kernel*, o que proporciona maior desempenho, quando comparado ao seu antecessor (*iptables*).

3.3. Traffic Control

O *Traffic Control* é um componente da plataforma de gerenciamento de pacotes em sistemas *Linux*, que provê um sistema sofisticado de controle e provisionamento de largura de banda nas transmissões. Este sistema suporta vários métodos envolvendo classificar, priorizar, compartilhar e limitar o tráfego de entrada e saída. O escalonamento de pacotes é a parte da pilha de rede do *kernel* que gerencia os *buffers* de transmissão e recepção de pacotes de todas as interfaces de rede (*NICs*). Os tratamentos relacionados à vazão de pacotes são definidos por *Qdiscs*, que possuem algoritmos para o gerenciamento das filas de entrada (*ingress*) e de saída (*egress*) dos dispositivos. Além disso, a classificação e a execução das políticas podem ser realizadas usando filtros.

3.4. XDP (eXpress Data Path)

O *XDP* fornece processamento de pacotes no nível mais baixo da pilha de software do sistema operacional. Deste modo, pode ser ideal para processamento de pacotes de alta velocidade sem sobrecarga ou comprometimento da capacidade de programação. O gancho *XDP* está localizado na cadeia de recebimento do controlador do dispositivo de rede antes da alocação dos buffers que carregarão o pacote pelas camadas do sistema operacional. Para isso, o driver de rede deve oferecer suporte a *XDP* caso contrário o programa será executado em modo genérico, mas com desempenho reduzido [HØILAND-JØRGENSEN et al. 2018].

XDP fornece um modelo de programação independente de protocolo, que permite escrever qualquer tipo de filtro que acessa os dados dos pacotes como uma sequência de bytes. Esse modelo permite uma versatilidade fundamental que não é alcançada com outros métodos de filtragem.

4. Estudo de caso

Os testes realizados por este trabalho buscam analisar as arquiteturas de filtragem e os consumos de energia associados num sistema Linux. Para a realização dos testes, foram conectados um computador *Raspberry PI* em um switch de rede *Gigabit Ethernet* com um computador gerador de carga em configuração *dual homed* como ilustrada na Figura 2 e um medidor energético *Yokogawa MW100 Data Acquisition Unit* [Standalone MW100 | Yokogawa Electric Corporation]. Um computador foi usado para gerar tráfego na rede e coletar os dados de consumo energético, enquanto o computador *Raspberry PI* é o *Device under Test (DuT)* que foi usado para processar os filtros de pacotes, assim como descrito no RFC [Hickman et al. 2003].

Quando se faz a análise da filtragem de pacotes, o *overhead* dominante é

computado em termos daqueles pacotes que são descartados pelos filtros. Ou seja, para avaliar a aplicação de regras, é relevante que a identificação dos pacotes resulte em seus descartes. Apenas no caso em que o projeto dos *buffers* não seja bem construído é que o custo dominante na aplicação de um filtro seria em termos dos pacotes que são aceitos [McCanne and Jacobson 1993]. Portanto, para as análises deste estudo, o custo energético e o consumo de recursos serão computados usando como base o processamento dos pacotes descartados.

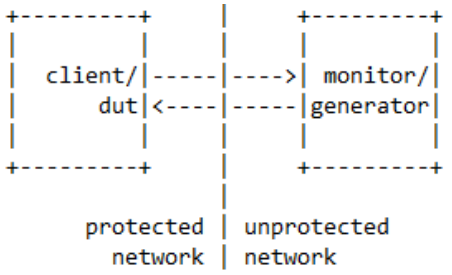

Figura 2. Configuração *Dual Homed* usada nos testes

Intuitivamente, esperaria-se que um menor consumo de energia deveria ser obtido pela filtragem de pacotes "no local", tão proximamente a onde são coletados, em vez de após copiá-los para algum outro buffer antes da filtragem. Assim, se o pacote não for aceito, apenas os bytes que foram necessários para o processo de filtragem são referenciados pelo Sistema Operacional. Deste modo, pode-se supor que aquelas arquiteturas que filtram em camadas mais baixas da pilha de protocolos do *kernel* tenham menor consumo de energia e menos uso de recursos computacionais para esta função.

5. Os experimentos

Os diferentes mecanismos de filtragem analisados foram avaliados em testes, cujas informações podem ser vistas em [SILVERIO, 2023]. Nos testes realizados, a máquina de origem do fluxo de carga foi configurada para gerar carga usando a ferramenta hping3. Para isso, foram aplicadas as seguintes configurações:

hping3 -I eth0 --udp --rand-source --destport 12345 -V -d $packet_size \
-i u1 192.168.0.79/24

A opção de gerar pacotes com IP fonte aleatório (--rand-source) busca causar um estresse na ferramenta *conntrack*. A ideia do teste é colocar o filtro simples em estresse máximo para fazer uma análise comparativa a partir da sua saturação.

Para todos os experimentos foi aplicado o mesmo conjunto de regras simples, com finalidade de analisar a sobrecarga quando um filtro é submetido a saturação:

- Descartar pacotes que sejam destinados ao endereço IP do DuT na porta 12345 provenientes de qualquer endereço de IP.

Os resultados foram obtidos através de ferramentas s*ysstat*, *ethtool*, *tcpdump* e *xpd-tools*, além das medições que foram realizadas no próprio filtro. A seguir está descrito como foram configuradas as regras para cada.

5.1. Descartando pacotes em espaço de usuário

Para os testes realizados, usou-se a vazão máxima obtida nos equipamentos utilizados, para que os pacotes pudessem ser analisados e ignorados (descartados) em espaço de usuário. Para esse teste foram aplicadas as seguintes regras e configurações:

iptables -I PREROUTING -t raw -d 192.168.0.79/24 -p udp --dport 12345 -j ACCEPT
iptables -I INPUT -t filter -d 192.168.0.79/24 -p udp --dport 12345 -j ACCEPT

Neste caso, a adição da regra na tabela *raw* (*-t raw*) tem o efeito de fazer com que o pacote seja aceito e siga para análise do mecanismo de controle de fluxo (*conntrack*). Se estiver ativo, o mecanismo *conntrack* irá atuar no ajuste da vazão deste fluxo. Já a 2ª regra, aceita a entrada do pacote destinado ao endereço IP da interface deste host.

5.2. Descartando pacotes em espaço de usuário, sem conntrack

Para os testes realizados, foram permitidos que os pacotes navegassem em sua vazão máxima, para que pudessem ser analisados e ignorados (descartados) em espaço de usuário. O *conntrack* é uma ferramenta em espaço de usuário que permite que o *kernel* verifique os fluxos de conexões e, assim, identifique todos os pacotes que compõem cada fluxo, para que possam ser tratados de maneira conjunta, e intenta tornar a pilha de rede do *kernel Linux statefull*. Para esse teste, foram aplicadas as seguintes regras:

iptables -I PREROUTING -t raw -d 192.168.0.79/24 -p udp --dport 12345 -j NOTRACK
iptables -I INPUT -t filter -d 192.168.0.79/24 -p udp --dport 12345 -j ACCEPT

Com as regras acima, desabilita-se o mecanismo de controle de fluxo e aceita-se a entrada do pacote especificado. Deste modo, qualquer bloqueio ou análise da entrada deste pacote deverá ocorrer em espaço de usuário (userspace).

5.3. cBPF (Berkeley Packet Filter on socket)

Nesse teste foi aplicado um filtro no formato BPF que descarta os pacotes no socket antes que sejam copiados para o espaço de usuário. Isso evita a sobrecarga (overhead) de ter que copiar dados do kernel para o espaço de usuário.

```
struct sock_filter code[] = {
        { 0x28, 0, 0, 0x0000000c }, { 0x15, 0, 10, 0x00000800 }, // pacote IPv4
        { 0x20, 0, 0, 0x0000001e }, { 0x15, 0, 8, 0xc0a8004f },    // 192.168.0.79
        { 0x30, 0, 0, 0x00000017 }, { 0x15, 0, 6, 0x00000011 },  // protocolo udp
        { 0x28, 0, 0, 0x00000014 }, { 0x45, 4, 0, 0x00001fff },
        { 0xb1, 0, 0, 0x0000000e }, { 0x48, 0, 0, 0x00000010 },
        { 0x15, 0, 1, 0x00003039 }, { 0x6, 0, 0, 0x00040000 },   // porta 12345
        { 0x6, 0, 0, 0x00000000 },
};
```

5.4. eBPF (Extended Berkeley Packet Filter on socket)

Nesse teste foi aplicado um filtro no formato *eBPF* que descarta os pacotes no socket

como feito no teste anterior.

```
0000000000000000 <socket_filter>:
   0:    71 12 17 00 00 00 00 00 r2 = *(u8 *)(r1 + 0x17)
   1:    55 02 05 00 11 00 00 00 if r2 != 0x11 goto +0x5 <LBB0_3>
   2:    61 12 1e 00 00 00 00 00 r2 = *(u32 *)(r1 + 0x1e)
   3:    55 02 03 00 c0 a8 00 4f if r2 != 0x4f00a8c0 goto +0x3 <LBB0_3>
   4:    b7 00 00 00 01 00 00 00 r0 = 0x1
   5:    69 11 24 00 00 00 00 00 r1 = *(u16 *)(r1 + 0x24)
   6:    15 01 01 00 30 39 00 00 if r1 == 0x3930 goto +0x1 <LBB0_4>
```

5.5. Netfilter (iptables input chain)

Nesse teste foi utilizada a ferramenta iptables para aplicar filtros nos ganchos de rede do kernel, e para esse caso, as regras foram configuradas para descartar os pacotes depois que fossem roteados pela pilha do sistema operacional. Foram aplicadas as seguintes regras:

```
iptables-legacy -I PREROUTING -t raw -d 192.168.0.79/24 -p udp --dport 12345 \
        -j NOTRACK
iptables-legacy -I INPUT -t filter -d 192.168.0.79/24 -p udp --dport 12345 -j DROP
```

Com essas regras, o pacote alvo (UDP porta 12345), passa pela tabela raw, sem estar sujeito ao mecanismo de controle de vazão. Já na 2a regra, o pacote é descartado.

5.6. Netfilter (iptables prerouting chain)

Nesse teste foi utilizada a ferramenta iptables para aplicar filtros nos ganchos de rede do kernel, e para esse caso as regras foram configuradas para descartar os pacotes antes que fossem roteados pela pilha do sistema operacional. Foram aplicadas as seguintes configurações:

```
iptables-legacy -I PREROUTING -t raw -d 192.168.0.79/24 -p udp --dport 12345 \
        -j DROP
```

Neste caso, o pacote é descartado antes de qualquer análise pelos mecanismos de controle de ingresso (INPUT) ou encaminhamento (FORWARD).

5.7. Netfilter (nftables)

Nesse teste foi utilizada a ferramenta nftables, sucessora do programa iptables, para aplicar filtros nos ganchos de rede do kernel. Para esse caso, as regras foram configuradas para descartar os pacotes antes que fossem roteados para a pilha do sistema operacional. Foram aplicadas as seguintes regras:

```
nft add table netdev filter
nft 'add chain netdev filter input \
        { type filter hook ingress device eth0 priority -500 ; policy accept ; }'
nft 'add rule netdev filter input ip daddr 192.168.0.79/24 udp dport 12345 counter drop'
```

5.8. eBPF (traffic control - tc)

Aqui, as configurações foram realizadas para descartar os pacotes aplicando o filtro no traffic control. Para isso. foram aplicadas as seguintes regras:

tc qdisc add dev eth0 ingress
tc filter add dev eth0 parent ffff: prio 4 protocol ip u32 match ip protocol 17 0xff match ip dport 12345 0xffff match ip dst 192.168.0.77/24 flowid 1:1 action drop

5.9. XDP (eXpress Data Path)

As regras foram configuradas para descartar os pacotes utilizando um filtro que usa a tecnologia xdp, e descarta os pacotes no espaço de driver:

xdp-filter load -m skb eth0
xdp-filter port -m dst 12345 -p udp
xdp-filter ip -m dst 192.168.0.79

6. Resultados Experimentais

A figura 3 apresenta os resultados obtidos com a monitoração da vazão do fluxo de saída e entrada de pacotes.

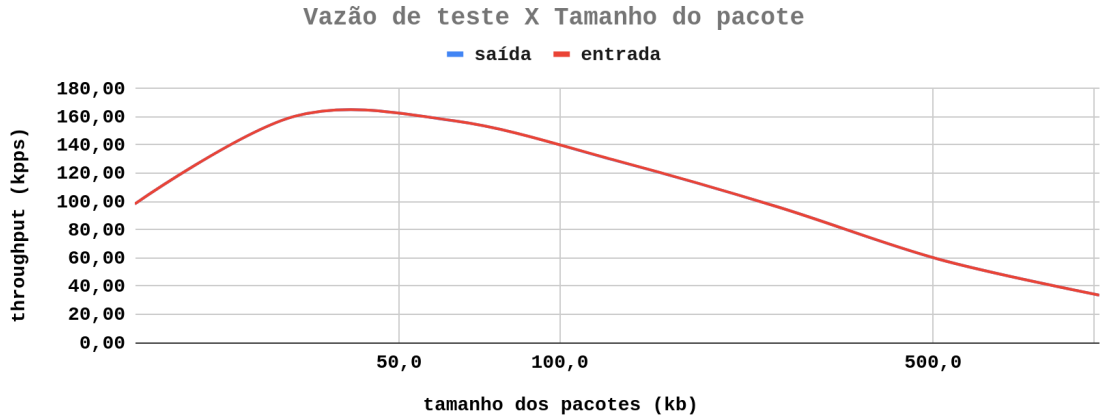

Figura 3. Vazão de entrada e saída X tamanho do pacote

É possível constatar que a entrada de pacotes no DuT (linha vermelha) sobrepõe a saída de pacotes no monitor (linha azul); ou seja, todos os pacotes enviados são recebidos sem descarte de processamento. Além disso, é observado um pico de vazão quando o tamanho do pacote é de 32 bytes. Deste modo, este foi o valor utilizado nos testes de energia.

A Figura 4 ilustra os resultados com relação ao consumo energético. O mecanismo *nftables* obteve os melhores resultados para processamento de pacotes, consumindo menos energia se comparado aos concorrentes para uma mesma carga. Para análise de pacotes, o *eBPF* obteve os melhores resultados, consumindo menos energia se comparado aos concorrentes para uma mesma carga. Os valores apresentados no eixo da energia (vertical) começam em 2.5 Wps, tendo em vista que este foi o consumo medido para o sistema computacional em estado ocioso.

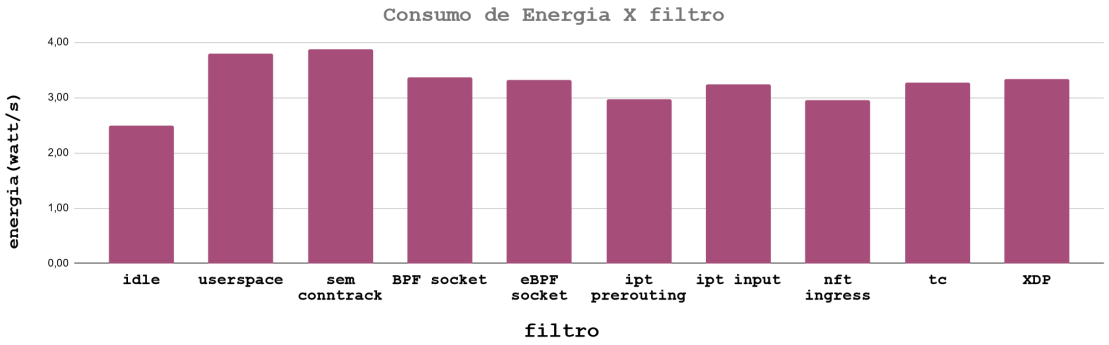

Figura 4. Gráfico Consumo de Energia X Tipo de Filtro

Considerando que quanto menos camadas o pacote atravessa na pilha do *kernel* potencialmente melhor é o desempenho obtido, seria de se esperar que *XDP* se saísse melhor. Para que isso acontecesse, contudo, seria necessário que este fosse carregado de forma nativa pelo *driver* (*offloaded*). Quando o programa é carregado na interface com modo genérico, o programa é executado depois da alocação da estrutura *xdp_buff*, que é criada a partir da estrutura *sk_buff* do pacote já no nível de *socket*. Ainda assim, é possível usufruir das funcionalidades de filtragem byte a byte que XDP oferece.

A Figura 5 ilustra os resultados com relação ao consumo de CPU. Novamente, a ferramenta *nftables* obteve melhores resultados para processamento de pacotes, enquanto *eBPF* foi mais eficiente para a análise de pacotes, consumindo menos recursos comparando-se com os concorrentes para uma mesma carga. Vê-se que tanto os filtros em *userspace*, quanto tc (*traffic control*) e *XDP* apresentaram consumo de CPU elevado.

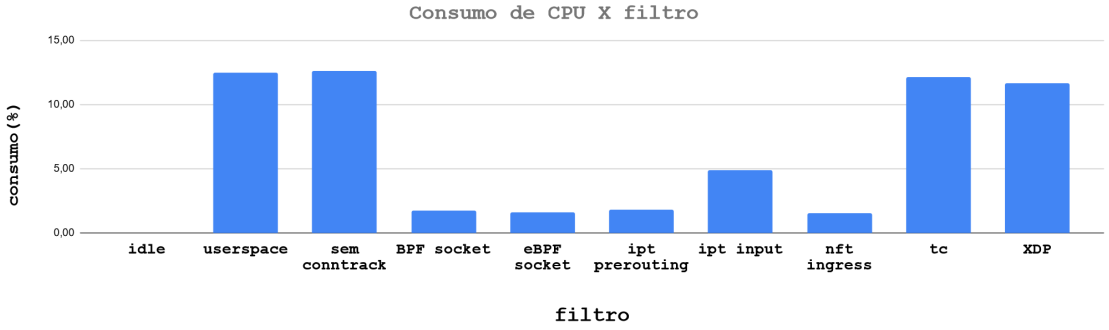

Figura 5. Gráfico do Consumo de CPU X Tipo de Filtro

Os valores obtidos pelos testes com *XDP* talvez não sejam tão intuitivos. Caso esse processamento ocorresse de fato na interface de rede, não haveria gasto de energia na *CPU*. Por outro lado, como ressaltado, o *driver* de rede da interface disponível não tinha este recurso nativo. Deste modo, o processamento das regras definidas ocorreu após ter sido feito todo o desencapsulamento dos campos dos protocolos envolvidos no processamento do pacote e o preenchimento de uma estrutura *xdp_buff* a partir de *sk_buff*.

A Tabela 1 mostra os valores de desvio padrão para os experimentos realizados, ilustrados nas figuras 4 e 5.

FILTRO	std (energia)	std (cpu)
idle	0,00	0,00
userspace	0,09	0,61
sem conntrack	0,07	0,86
BPF socket	0,03	0,37
eBPF socket	0,03	0,27
ipt prerouting	0,04	0,18
ipt input	0,06	0,30
nft ingress	0,05	0,28
tc	0,04	3,35
XDP	0,13	0,43

Tabela 1. Desvios padrão para os experimentos realizados

Nos testes relatados nas Figuras 4 e 5, utilizou-se pacotes de tamanho 32 bytes, que foram os que possibilitaram a maior vazão nas comunicações avaliadas. Para avaliar o efeito do tamanho dos pacotes nas medições de energia, contudo, novos testes foram realizados, com valores de 16 bytes a 1472 (MTU Eth - cabeçalho IPv4 - cabeçalho UDP). Um aspecto relevante sobre esses testes, contudo, é que a vazão obtida decaiu com pacotes maiores. Os resultados obtidos são apresentados na Figura 6. Embora os gastos de energia foram menores para pacotes maiores, isso deveu-se ao menor número de pacotes processados nesses casos. A proporção de gasto de energia entre os mecanismos, contudo, manteve-se praticamente o mesmo em todos os casos.

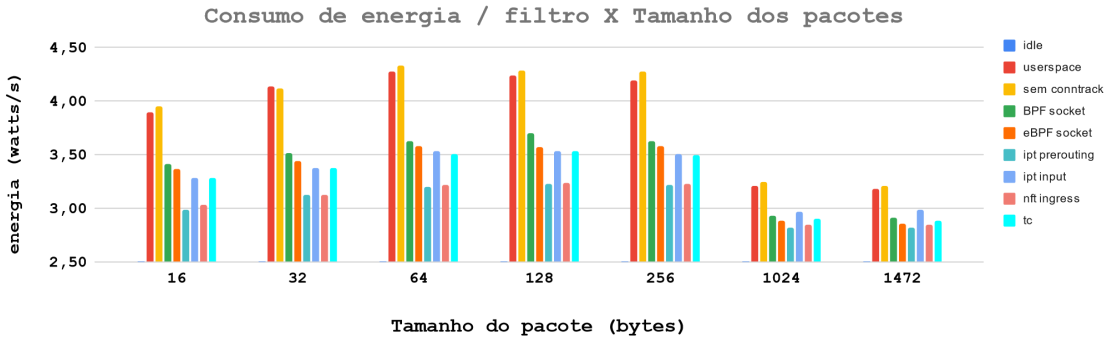

Figura 6. Gráfico do Consumo de energia / Filtro X Tamanho dos pacotes

7. Conclusões

Para efeitos de comparação, tratar filtros que aplicam regras sempre nas camadas mais baixas da pilha de rede é mais eficiente e consome menos recursos de CPU e energia do que as técnicas que fazem o pacote atravessar um maior número de camadas na pilha do sistema operacional. Sob a óptica dos dispositivos de menor capacidade, técnicas que utilizam *eBPF* na análise de pacote são aplicáveis; ainda o *nftables* tem eficiência ligeiramente melhor que o sucessor iptables para o processamento de pacotes.

Esperava-se que, com relação ao consumo energético, *XDP* fosse mais eficiente que as outras técnicas, pois analisa pacote e o descarta antes mesmo que este escale no processamento feito pelas camadas superiores da pilha de protocolos. Contudo, o modo nativo não é suportado na maioria dos *NICs* e em dispositivos *low-end*, o que também ocorre com o computador *Raspberry Pi* utilizado neste estudo. Um caminho seria trabalhar para que isso seja amplamente aplicado em projetos abertos para sistemas

low-end, já que essa técnica é a mais eficiente para tratar protocolos independentes.

Com relação à vazão obtida nos testes, novas avaliações devem ser feitas utilizando maiores volumes de dados e diferentes padrões de acesso.

Agradecimentos

Esse artigo apresenta uma parte dos testes e estudos desenvolvidos no trabalho de iniciação científica PIBIC 2022-2023 e não teria sido possível sem o apoio do CNPQ (Conselho Nacional de Desenvolvimento Científico e Tecnológico).

Referências

HICKMAN, B. et al. **Benchmarking Methodology for Firewall Performance**. Disponível em: <https://datatracker.ietf.org/doc/html/rfc3511>. Acesso em: 28 ago. 2023.

HØILAND-JØRGENSEN, T. et al. The eXpress data path. **Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies**, 4 dez. 2018.

JIANG, W. et al. The Road Towards 6G: A Comprehensive Survey. **IEEE Open Journal of the Communications Society**, v. 2, p. 334–366, 2021.

MCCANNE, S.; JACOBSON, V. The BSD packet filter: a new architecture for user-level packet capture. p. 2–2, 25 jan. 1993.

MELKOV, D.; SALTIS, A.; PAULIKAS, S. Performance Testing of Linux Firewalls. **2020 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream)**, abr. 2020.

MITRA, P. et al. **Towards 6G Communications: Architecture, Challenges, and Future Directions**. Disponível em: <https://ieeexplore.ieee.org/document/9580084>. Acesso em: 28 ago. 2023.

PORAMBAGE, P. et al. The Roadmap to 6G Security and Privacy. **IEEE Open Journal of the Communications Society**, v. 2, p. 1094–1122, 2021.

SCHOLZ, D. et al. Performance Implications of Packet Filtering with Linux eBPF. **2018 30th International Teletraffic Congress (ITC 30)**, set. 2018.

SILVERIO, A. **Energy Measurement Packet Filtering Analysis**. Disponível em: <https://github.com/arthunix/Linux-Packet-Filtering-Energy-Analysis>. Acesso em: 28 ago. 2023.

Standalone MW100 | Yokogawa Electric Corporation. Disponível em: <https://www.yokogawa.com/solutions/discontinued/standalone-mw100/#Details__Features__Smart-Logging>. Acesso em: 28 ago. 2023.

VIEIRA, M. A. M. et al. Fast Packet Processing with eBPF and XDP. **ACM Computing Surveys**, v. 53, n. 1, p. 1–36, 29 maio 2020.

XIAO, Y. et al. Edge Computing Security: State of the Art and Challenges. **Proceedings of the IEEE**, v. 107, n. 8, p. 1608–1631, ago. 2019.