Sistema de Detecção de Intrusão *Serverless* em uma SmartNIC

Lucas F. S. Duarte[1], **Racyus D. G. Pacífico**[2], **Marcos A. M. Vieira**[2], **José A. M. Nacif**[1]

[1]Universidade Federal de Viçosa (UFV) – Florestal, MG – Brasil

[2]Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte, MG – Brasil

{lucas.f.duarte,jnacif}@ufv.br, {racyus,mmvieira}@dcc.ufmg.br

Resumo. *Segurança de dados tornou-se um fator crucial no contexto de redes de computadores. Entre 2015 e 2020, estima-se que o prejuízo causado pelo roubo de informações na Internet seja de R$ 26 trilhões. Nesse cenário, Sistemas de Detecção de Intrusão (SDIs) são elementos essenciais na busca de ameaças. Contudo, SDIs são instalados em máquinas convencionais com pouco poder de processamento e altos custos operacionais. Offloading de funções de rede em SmartNICs e o paradigma Serverless surgem como soluções eficientes, pois combinam flexibilidade e programabilidade com poder de processamento de pacotes em hardware. Nesse trabalho propomos um SDI Serverless com offloading de filtros definidos pelo usuário na SmartNIC Netronome. Foram criados diferentes filtros usando a tecnologia eBPF para validar o sistema. Os resultados obtidos mostram que ataques são detectados em taxa de linha.*

1. Introdução

Segurança de dados tornou-se um fator crucial no contexto de redes de computadores. Entre 2015 e 2020, estima-se que o prejuízo causado pelo roubo de informações na Internet foi de aproximadamente R$ 26 trilhões [Digital 2017]. Projeções realizadas em 2019 indicam que os prejuízos sofridos por empresas através de cibercrimes e falhas em segurança excederão o valor de 5 trilhões de dólares em 2024 [Juniper 2019]. Além disso, vazamentos de dados ocorridos no ano de 2019 e descobertos recentemente em 2021 expuseram dados sensíveis de cerca de 220 milhões de brasileiros e geraram uma série de prejuízos pessoais, por exemplo, clonagem de cartões de crédito e criação de empresas fantasmas [NH 2021]. Nesse cenário é importante manter as informações trafegadas na rede de forma segura e protegida. Sistemas de Detecção de Intrusão (SDIs) são elementos essenciais neste contexto, pois permitem que o conteúdo trafegado na rede seja analisado em busca de possíveis ameaças, vulnerabilidades e ataques que possam comprometer a integridade do sistema e permitir que informações sejam obtidas através de meios ilegais.

SDIs são executados em servidores ou computadores convencionais com CPUs x86_x64. Essa abordagem não é eficiente e pode gerar sobrecarga de recursos, uma vez que o processamento não é exclusivo, aumentando o custo operacional. Processamento do tráfego de pacotes utilizando *offloading* em hardware aumenta a eficiência no processamento, permitindo que pacotes sejam analisados com alta vazão e baixa latência. SmartNICs surgem como hardware alternativo pois suportam *offloading* de funções. Além disso, elas contêm vários núcleos de processamento e consomem pouca energia.

O modelo de arquitetura *Serverless* permite que trechos de códigos de funções de rede específicas sejam executadas em um provedor de computação em nuvem com

recursos alocados dinamicamente e sob demanda. Os custos dos serviços fornecidos por um provedor *Serverless* são calculados considerando somente os recursos utilizados durante as operações. Em geral, o código é escrito em forma de funções e encapsulado em contêineres que podem ser ativados por diversos eventos, como a chegada de requisições HTTP. Os usuários neste modelo se preocupam apenas com a implementação das funções, deixando a parte de execução das funções e infraestrutura a cargo do provedor. Este modelo é conhecido por ser escalável e flexível. Vários provedores de serviços em nuvem, tais como, Amazon AWS [Services 2017], Microsoft Azure [Microsoft 2017] e Google Cloud [Google 2017]) oferecem suporte ao modelo *Serverless*.

Neste trabalho é proposto um SDI *Serverless* com *offloading* de funções (filtros) em hardware capaz de processar pacotes com flexibilidade e programabilidade usando diferentes filtros definidos pelo usuário. Nosso sistema une a eficiência energética de uma SmartNIC à flexibilidade e escalabilidade provida pelo arcabouço *Serverless* OpenFaaS. Ele também permite que funções de filtragem e detecção de ameaças sejam instaladas e executem com *offloading* em hardware sem perdas de pacotes. Usuários podem criar filtros escritos na linguagem C utilizando eBPF (*extend Berkeley Packet Filter*) para analisar pacotes de forma programável e adicioná-los facilmente ao conjunto de funções *Serverless* disponível em nosso SDI. Nós usamos a SmartNIC Netronome Agilio CX 2x10 GbE para realizar o *offloading* de funções e processamento de pacotes.

As principais contribuições deste trabalho são: (i) *offloading* de filtros do SDI em hardware provendo flexibilidade e programabilidade usando programas eBPF via OpenFaaS; (ii) integrar um conjunto de filtros de inspeção do conteúdo de pacotes usando expressões regulares em um sistema de nuvem; (iii) agregar no SDI vários tipos de tecnologias para fornecer alto poder de processamento em hardware com escalabilidade de funções em software e eficiência energética. Todos esses *insights* contribuem para a redução do custo operacional. Contornamos diversos desafios, como produzir filtros eBPF/XDP que percorrem o conteúdo do pacote usando instruções de retorno, por exemplo, `goto`. Além disso, o SDI suporta a reprogramação em tempo de execução com escalabilidade de funções sem perda de pacotes durante o agendamento de funções entre software e hardware.

O restante deste artigo está organizado da seguinte forma: Na seção 2 é apresentada a visão geral do sistema. A seção 3 aborda detalhes referentes à implementação do SDI. A avaliação e resultados do SDI são discutidos na seção 4. Na seção 5 apresentamos a limitação do sistema. Em seguida, os trabalhos relacionados são discutidos na seção 6. Por fim, na seção 7 conclusões e trabalhos futuros são descritos.

2. Visão geral do sistema

O sistema proposto é dividido em duas partes: o componente *Serverless* e o dispositivo de hardware. A primeira parte é composta por uma interface de linha de comando, o eBPFaaS CLI, que possui comandos para facilitar o processo de criação, execução e remoção de filtros, e pelo arcabouço *Serverless* OpenFaaS. Esses componentes são responsáveis pela interação com o usuário do sistema durante a criação e o gerenciamento de filtros de processamento de pacotes. Usuários podem criar filtros com suporte a mecanismos de reconhecimento de padrões como expressões regulares e máquinas de

estado finito em C utilizando eBPF. Além disso, o *ebpfaas-cli* pode convertê-los em filtros *serverless* encapsuladas em contêineres.

Uma vez criado, o filtro fica disponível no sistema para ser utilizado de forma escalável. Usuários podem carregar filtros no hardware invocando-os através de requisições HTTP enviadas através do *ebfaas-cli*. O OpenFaaS recebe as requisições e inicia o processamento no contêiner, que compila o filtro em instruções eBPF e as envia para a fila de espera. A fila armazena os filtros utilizando o modelo FIFO (*first-in, first-out*). A segunda parte do sistema é composta pelo hardware que recebe filtros da fila de acordo com os parâmetros de configuração estabelecidos pelo usuário. Esses parâmetros permitem configurar o tempo de execução de cada filtro na instância física. Informações sobre o resultado da operação são transmitidas através de respostas HTTP ao usuário no final do processamento. Após isso, o contêiner e o hardware são liberados para serem utilizados. A figura 1 apresenta uma visão geral dos componentes que compõem o sistema.

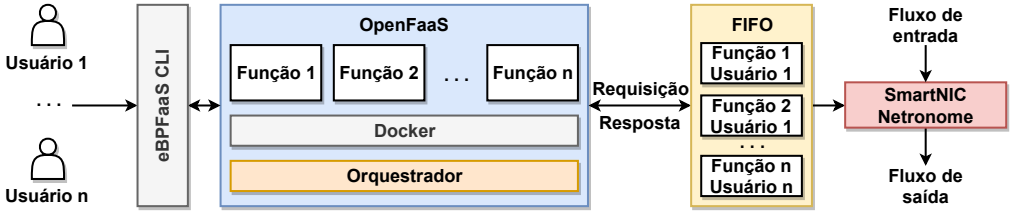

Figura 1. Visão geral do sistema.

2.1. eBPF e XDP

eBPF (*extended Berkeley Packet Filter*) [eBPF 2014, Vieira et al. 2019, Vieira et al. 2020b] e XDP (*eXpress Data Path*) [IO Visor 2016] são tecnologias utilizadas no processamento rápido de pacotes da rede. O eBPF é utilizado para escrever programas executados no espaço do usuário que agem como filtros capazes de analisar o tráfego de pacotes da rede, adicionando programabilidade no processamento de informações. O eBPF está disponível no *kernel* do Linux desde a versão 3.15. Programas eBPF podem ser escritos utilizando outros tipos de linguagens, tais como, C e P4.

O XDP é um caminho de dados compatível com eBPF disponível no *kernel* do Linux desde a versão 4.8. O XDP adiciona um gancho no caminho de entrada de pacotes do *kernel* para criar uma relação entre o fluxo de dados e um programa eBPF fornecido pelo usuário, permitindo que ele decida o destino do pacote. Além disso, o XDP atua no ponto mais baixo da pilha de rede, permitindo que o processamento seja realizado de forma rápida sem comprometer a programabilidade. Ele também permite que funções sejam implementadas dinamicamente sem exigir modificações no *kernel*. O XDP também permite que a computação seja transferida para a placa de rede, utilizando *offloading* de funções de modo dinâmico. O SDI proposto utiliza as tecnologias eBPF e XDP para criação de filtros programáveis para serem carregados e executados em hardware.

2.2. Arquitetura *Serverless*

O modelo de execução FaaS (*Function as a Service*) permite que funções possam gerenciar a lógica e o estado da aplicação em um servidor através de serviços. Os

sistemas que utilizam FaaS são orientados a eventos, isto é, têm seu funcionamento acionado através da detecção de eventos de ação como requisições HTTP, execuções agendadas, entre outros [Fox et al. 2017]. Para isso, contêineres *stateless* que não armazenam informações sobre o que foi realizado em cada execução (como arquivos abertos, novos valores de variáveis de ambiente, etc) são utilizados como unidade de execução.Contêineres são ambientes de trabalho com usuários sistemas de arquivos, processos e pilha de rede próprios não compartilhados com o sistema operacional.

O modelo de arquitetura *Serverless* é baseado no modelo de execução FaaS e permite que trechos de códigos de ação específica sejam executados em um provedor de computação em nuvem com recursos alocados dinamicamente e sob demanda. Os custos são calculados considerando somente os recursos utilizados durante as operações. Em geral, o código é escrito no formato de funções e é encapsulado em contêineres que podem ser ativados por diversos eventos, como a chegada de requisições HTTP. O desenvolvimento e a implantação das funções são as únicas responsabilidades dos usuários. O modelo *Serverless* é conhecido por ser escalável e flexível [Vieira et al. 2020a].

OpenFaaS [Ellis 2016] é um arcabouço *Serverless* de código aberto que facilita o processo de criação, implantação e gerenciamento de funções de forma escalável. Esse arcabouço utiliza contêineres para encapsular funções e orquestradores conhecidos pela comunidade da área para prover escalabilidade, como Docker Swarm [Docker Inc. 2014] e Kubernetes [Google Inc. 2015]. Um orquestrador é uma ferramenta responsável pela automatização, implantação, gerenciamento e escalabilidade de um sistema de contêineres. Ele permite que contêineres sejam instalados em diferentes computadores como se fosse um único dispositivo através da criação de *clusters*. OpenFaaS permite que funções sejam escritas em qualquer linguagem de programação e pode ser utilizado através de interfaces gráficas ou através de linha de comando com o auxílio de programas que permitem o envio e o recebimento de mensagens HTTP. Além disso, o OpenFaaS disponibiliza uma API (*Application Programming Interface*) que permite que outros componentes sejam facilmente acoplados ao arcabouço, por exemplo, o componente Prometheus, responsável pelo monitoramento de dados do OpenFaaS.

2.3. Comunicação e gerenciamento

A comunicação entre as funções *Serverless* e a fila de filtros eBPF/XDP é realizada via soquetes TCP/IP implementados em Python. A figura 2 apresenta o modelo de comunicação e gerenciamento do SDI.

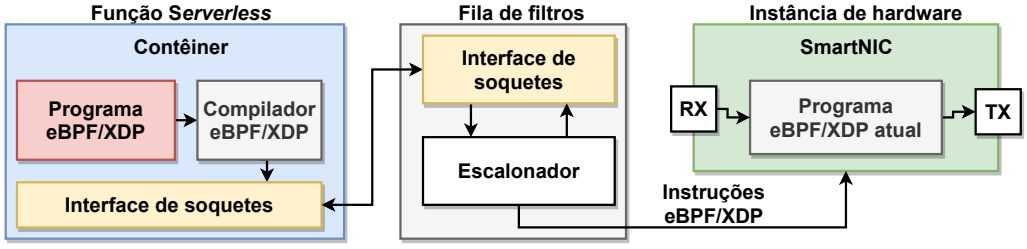

Figura 2. Visão geral comunicação entre funções e fila de filtros eBPF/XDP.

A fila de filtros atua como um escalonador e é responsável por armazenar os programas eBPF/XDP recebidos através das solicitações de execução realizadas pelos

usuários. Ela recebe os conjuntos de instruções e faz o *offloading* das instruções no hardware na ordem em que foram recebidos, de acordo com o tempo de duração da execução parametrizado pelo usuário. O programa a ser executado é enviado para a SmartNIC e executado de acordo com o tempo definido pelo usuário. Após o programa ser carregado na SmartNIC, o hardware passa a filtrar pacotes de entrada e produzir o fluxo de saída de pacotes de acordo com o comportamento e regras presentes no programa atual. O processamento termina quando o tempo de execução definido se esgotar, liberando o hardware para ser utilizado pela próxima função na fila de execução.

2.4. Processamento de pacotes em SmartNICs

Interfaces de rede inteligentes, ou SmartNICs, são dispositivos que proveem programabilidade e flexibilidade nas interfaces de rede. Essas características permitem executar tarefas que NICs (*Network Interface Cards*) tradicionais não são capazes de realizar, tais como, *offloading* de funções [Liu et al. 2019a, Pacífico et al. 2021], acelaração de pacotes TCP [Moon et al. 2020, Arashloo et al. 2020], e balanceamento de carga [Miao et al. 2017].

O SDI proposto realiza o processamento de pacotes sobre a SmartNIC Agilio Netronome CX2x10GbE. Netronome tem 60 núcleos de processamento de fluxo programáveis e 40 núcleos de processamento de pacotes, além de duas portas CX2 que suportam a transmissão de dados em uma taxa de até 10 Gbps. Netronome é uma SmartNIC que contém alto poder de processamento de pacotes com baixo custo energético, que pode ser facilmente adaptado a demanda da rede e operadores [Netronome 2021]. Nós escolhemos a SmartNIC Netronome para compor o sistema porque outros fabricantes de SmartNIC, tais como Mellanox [Nvidia 2021] e Intel [Intel 2021] não suportam o *offloading* de programas eBPF.

3. Implementação

Nesta seção descrevemos detalhes de implementação dos componentes do sistema. A interface entre o usuário e a SmartNIC foi implementada a partir do arcabouço OpenFaaS [Ellis 2016] em conjunto com o orquestrador Docker Swarm [Docker Inc. 2014]. O orquestrador é o software responsável por automatizar os processos de implantação e gerenciamento de contêineres. OpenFaaS é compatível com grandes orquestradores presentes no mercado, tal como, Kubernetes [Google Inc. 2015]. Entretanto, escolhemos o Docker Swarm como orquestrador devido à facilidade e o bom desempenho em clusters com poucos contêineres. Implementamos uma série de funções de operação do SDI para que o usuário possa ter total controle sobre os contêineres que armazenam os filtros do sistema.

3.1. Interface *ebpfaas-cli*

Para facilitar a interação do usuário com o sistema, criamos um programa CLI (*Command Line Input*) denominado *ebpfaas-cli* que conta com quatro operações: criar, atualizar, remover, e executar um filtro eBPF/XDP. A operação `criar` requer o nome do arquivo do filtro como parâmetro. Esta operação é responsável por criar e implantar um novo contêiner no sistema de acordo com o filtro escolhido.

O contêiner é criado a partir de um *template*, que são conjuntos de arquivos utilizados como modelo para a criação de funções *Serverless* no OpenFaaS. O OpenFaaS

possui um repositório oficial composto por *templates* [Ellis 2017] para a criação de funções escritas em várias linguagens, tais como, Javascript, Go, Python e Java. Todos os *templates* são formados por um programa *index* que atua como um ponto de entrada e saída de dados, um programa *handler* que realiza o processamento, e um arquivo *Dockerfile* contendo regras para criar uma imagem Docker, responsável pela origem do contêiner do filtro. Esses arquivos podem ser alterados pelo usuário para suportar qualquer linguagem de programação.

Para o SDI proposto criamos um novo *template c-ebpf* [Duarte 2021] baseado no Alpine Linux, uma versão do sistema operacional que possui apenas 2 MB. Nesse *template* instalamos um conjunto de pacotes, por exemplo, o compilador `Clang`. Além disso, atualizamos os arquivos de cabeçalhos do *Kernel* para que o *template* suporte a linguagem C e tecnologias eBPF/XDP. Disponibilizamos os arquivos necessários para se criar contêineres compatíveis com C e eBPF/XDP no Docker Hub [Docker 2021], uma plataforma de compartilhamento de imagens Docker.

A operação `atualizar` é usada para atualizar o contêiner implantado ou o código do filtro armazenado no contêiner. A terceira operação (`remover`) é responsável por excluir um contêiner do sistema, removendo o filtro da lista de opções disponíveis. Finalmente, a operação `executar` refere-se a operação de execução do filtro. Nesta operação, o usuário precisa fornecer dois parâmetros: o nome do filtro desejado e o tempo de execução do filtro no hardware. Ambos parâmetros fornecidos são encapsulados em uma requisição HTTP que é enviada ao componente de interface do OpenFaaS, que inicializa o contêiner correspondente à função solicitada pelo usuário. O parâmetro tempo de execução é utilizado pelo escalonador da fila de filtros para determinar por quanto tempo o filtro será executado no hardware. Essa operação informa ao usuário se o filtro foi adicionado à fila de filtros.

O *ebpfaas-cli* também contêm comandos auxiliares que possibilitam os usuários inicializar ou atualizar o ambiente de desenvolvimento *Serverless* rapidamente, usando o comando `build`. Esse comando une as operações de inicialização do cluster do Docker Swarm, a instalação/implantação de todos os contêineres dos componentes do OpenFaaS, e a geração de credenciais válidas de acesso ao sistema.

3.2. Componentes de comunicação

Cada contêiner criado no sistema é composto pelo programa *index*, programa *handler* e filtro eBPF/XDP. Quando o contêiner é executado, o programa *index* envia o tempo de execução do filtro recebido, como parâmetro para o programa *handler* que inicia o processamento. Esse programa é responsável por compilar e gerar as instruções eBPF a partir do programa escrito em C, e enviá-las para o componente de transmissão. O componente de transmissão recebe as instruções eBPF e encaminha as respectivas instruções para a fila de filtros. Todo o procedimento de transmissão ocorre via soquetes TCP/IP.

3.3. Fila de filtros eBPF/XDP

É responsável por receber as instruções eBPF/XDP e armazená-las em um *buffer* que funciona de acordo com o algoritmo de escalonamento FIFO (*First In, First Out*). Esse *buffer* atua como um escalonador, determinando qual programa será carregado

na SmartNIC. O primeiro programa a chegar na fila é imediatamente carregado no dispositivo e executado de acordo com o período de tempo definido pelo usuário. O sistema informa ao usuário sobre o estado da operação de carregamento do programa no *buffer*, imediatamente após sua realização. Após esse procedimento, todos os programas recebidos posteriormente são armazenados até que o hardware esteja disponível para processar o próximo filtro.

4. Avaliação

Nesta seção apresentamos a avaliação dos experimentos do sistema. Avaliamos três tipos de experimentos: (i) vazão de pacotes de acordo com o filtro carregado; (ii) latência (tempo gasto) ao fazer uma requisição, *offloading* no dispositivo e resposta do término de processamento do filtro e (iii) número de requisições que podem ser realizadas simultaneamente.

4.1. Topologia experimentos

A topologia dos experimentos (figura 3) foi composta por dois computadores (`computador A e B`). O `computador A` contém a SmartNIC Intel X710 DA-2 que executa o gerador de tráfego (`pktgen-DPDK`) [Turull et al. 2016], para produzir o fluxo de pacotes que alimenta o sistema. O gerador de tráfego envia pacotes com tamanhos que variam entre 64 e 1500 bytes em taxa de linha de 10 Gbps. Além disso, neste computador OpenFaaS e interface de usuário *ebpfaas-cli*. No `computador B` ocorre todo o processamento e análise dos pacotes com e sem ataques. Neste computador a SmartNic Agilio CX 2x10 GbE da fabricante Netronome está acoplado. Essa SmartNIC contêm um barramento PCI *express* (PCIe) da terceira geração, duas interfaces SFP+ de 10 GbE, uma memória RAM DDR3 com capacidade de armazenamento de 2 GB, e um processador de pacotes Netronome NFP-4000 com múltiplos núcleos.

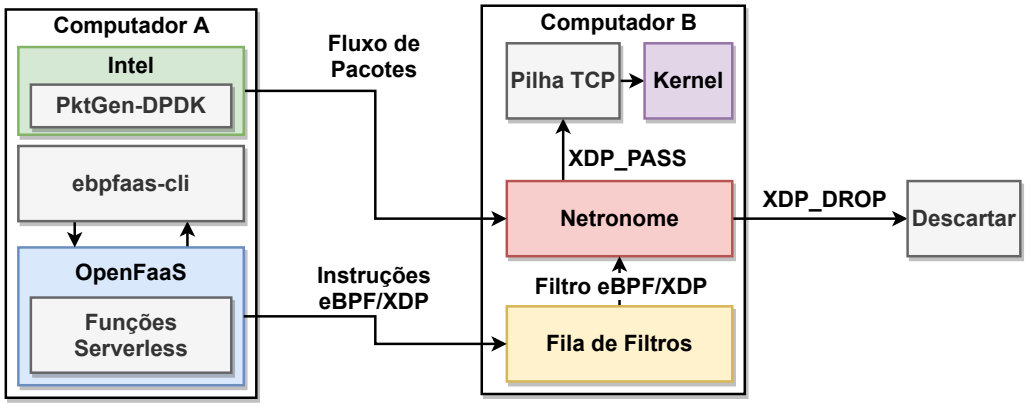

Figura 3. Escopo topologia experimentos.

Pacotes recebidos na Netronome são classificados de acordo com o filtro atual carregado. Se o pacote for classificado sem ataque, a ação XDP_PASS é aplicada para que o pacote seja enviado para a pilha TCP, e posteriormente para o *Kernel*. Se o pacote for classificado com ataque, a ação XDP_DROP é aplicada e o pacote é descartado.

Tabela 1. Filtros eBPF/XDP implementados no sistema.

Filtros	Nº de Linhas em C	Nº de Instruções	Utiliza ERs	Nº de Estados
Ações no pacote	4	2	Não	0
Mitigação DDoS	57	30	Não	0
BitTorrent	103	193	Sim	7
Injeção SQL (Tautologia)	112	771	Sim	5
Injeção SQL (*Sleep*)	114	1073	Sim	4
Malware	115	1255	Sim	3

4.2. Casos de uso

Para avaliar a capacidade do sistema de impedir ataques e tentativas de invasão com *offloading* de filtros em hardware e integração do OpenFaaS ao SDI, implementamos seis filtros apresentados na tabela 1. Nessa tabela demonstramos o número de instruções, o número de linhas, o número de estados de cada filtro, e se o filtro utiliza expressões regulares (ERs). A seguir, apresentamos uma breve descrição de cada filtro implementado:

Ações no pacote: Esse filtro foi utilizado como *baseline* para testes do SDI. Neste filtro duas ações no pacote do XDP (XDP_PASS e XDP_DROP) foram utilizados. O XDP_PASS faz com que o processador envie os pacotes recebidos para a pilha de rede. O XDP_DROP descarta todos os pacotes recebidos na interface da Netronome, impedido que os pacotes sejam encaminhados para a pilha de rede.

Mitigação DDoS (*Distributed Denial of Service*): É caracterizado por tentar saturar a banda ou sobrecarregar os recursos computacionais dos equipamentos, limitando a capacidade ou tornando indisponível o serviço, servidor ou rede alvo. Esse filtro analisa o fluxo da rede em busca de pacotes com características do ataque DDoS, impedindo de terem sucesso no ataque e acesso à rede [Bertin 2017].

Injeção SQL (Tautologia): O ataque de injeção SQL é caracterizado pela inserção de trechos de código SQL em sistemas com bases de dados vulneráveis na tentativa de obter acesso ou causar danos à integridade das informações armazenadas. Uma forma de realizar esse ataque é através da injeção de códigos que produzam tautologias. Por exemplo, se o sistema possui a *query* SELECT * FROM Clientes WHERE cliente_cpf = “cpf” acessível através de um campo de texto sem validação. Usuários mal intencionados podem inserir a sequência de caracteres “OR 1 = 1 como valor para o parâmetro *cpf*. A *query* será então executada como SELECT * FROM Clientes WHERE cliente_cpf = “” OR 1 = 1. Como a expressão *1 = 1* é sempre válida, todas as linhas da tabela *Clientes* serão exibidas como resultado, expondo dados potencialmente sensíveis. Neste filtro uma expressão regular para detectar ataques que utilizam tautologias foram escritas na linguagem SQL.

Injeção SQL (*Sleep*): Esse filtro detecta uma variante do ataque de injeção SQL, e inclui uma *query* com a chamada da função *sleep* do SQL no campo *User-Agent* presente em requisições HTTP. Essa função causa atraso em segundos na execução de operações SQL. Como *queries* são executadas de forma sequencial, qualquer outra requisição recebida pelo sistema será apenas respondida após o término da operação com o atraso causado pela função *sleep*. Esse atraso pode expor vulnerabilidades do

sistema. Essa informação pode ser utilizada por usuários mal intencionados para causar indisponibilidade, lentidão ou ser porta de entrada para outros tipos de ataques.

Filtro BitTorrent: BitTorrent é um protocolo de compartilhamento de arquivos P2P (*peer-to-peer*) utilizado na distribuição de dados sem que o transmissor pague por recursos em hardware, armazenamento e banda ao utilizar clientes com acesso a arquivos como pontos de transmissão. Pacotes BitTorrent na rede podem causar sobrecarga da rede, gerando lentidão e aumento dos custos operacionais. Este filtro detecta quatro tipos de pacotes BitTorrent e impede que esses pacotes sejam trafegados e transmitidos na rede.

Filtro Malware: Detecta a tentativa de inserção de códigos maliciosos em ambientes com a versão 7 do PHP, presente em muitos sistemas distribuídos. A inserção dos códigos acontece via chamadas da função *assert*, que é capaz de executar qualquer código PHP fornecido como parâmetro na forma de sequência de caracteres. Como não existe nenhuma restrição sobre a natureza do código executado, usuários mal intencionados tem alto nível de flexibilidade em seus ataques. Isso permite que obtenham e manipulem qualquer informação presente no sistema. Além de detectar a presença da função *assert* no conteúdo do pacote, esse filtro também detecta chamadas da função *base64_decode*, utilizada para manter oculta a sintaxe da linguagem durante a transmissão, dificultando sua detecção.

4.3. Resultados

Os experimentos realizados avaliaram a vazão da Netronome executando os filtros eBPF/XDP, latência referente ao tempo gasto na realização de requisições e carregamento dos filtros, e taxa de requisições máxima de acordo com o filtro carregado.

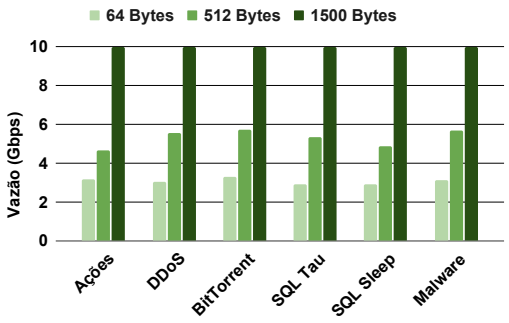

Figura 4. Vazão.

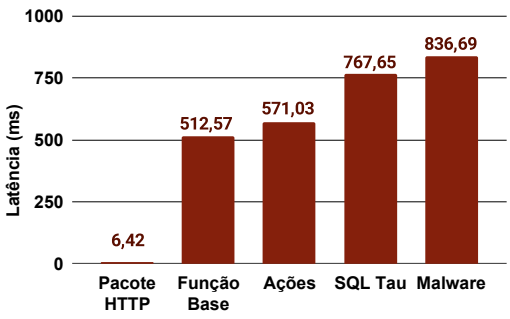

Figura 5. Latência.

Vazão: a figura 4 apresenta a vazão média para pacotes de 64, 512 e 1500 bytes da placa Netronome, executando os filtros eBPF/XDP apresentados na tabela 1. Neste experimento pacotes de 1500 bytes alcançaram a vazão máxima de 10 Gbps. No entanto, pacotes de 64 e 512 bytes tiveram a vazão afetada devido ao aumento no número de pacotes processados. Mesmo com a vazão reduzida para pacotes de 64 e 512 bytes ter poder de processamento de pacotes em hardware ainda é viável em relação a abordagens tradicionais e redução de custos operacionais.

Latência: a figura 5 apresenta a latência média observada ao realizar requisições para três filtros apresentados na tabela 1. Escolhemos os filtros de acordo com o menor, médio e maior número de instruções. A latência média obtida para os filtros

representa o tempo de transmissão das instruções via soquetes de dados e componentes de comunicação do OpenFaaS. Além disso, comparamos os valores da latência dos filtros com dois casos de teste base: o envio de um pacote HTTP sem o OpenFaaS e o carregamento de uma função base que não envia nenhuma instrução. Os resultados deste experimento demonstram que a latência imposta pela comunicação cresce de forma proporcional de acordo com o aumento na quantidade de instruções. O tempo de comunicação entre os componentes internos do OpenFaaS representa cerca de 98,7% do tempo de envio de informações em comparação com o envio do pacote HTTP.

Taxa de requisições: realizamos múltiplas requisições dos filtros de forma sequencial, sem intervalos, durante um período de um minuto com o objetivo de medir a taxa de requisições simultânea que pode ser realizada no SDI de acordo com o filtro. A figura 6 apresenta a taxa de requisições para cada filtro. A taxa de requisições é medida em requisições por segundo e representa a média do número máximo de requisições realizadas com sucesso durante um minuto. O experimento com cada filtro foi repetido 33 vezes para obter a média. Os resultados obtidos mostram que a quantidade de requisições é inversamente proporcional ao número de instruções do filtro. Esse resultado indica que quanto maior é o número de instruções do filtro, maior é o tempo de carregamento, e menor é o número máximo de requisições por segundo. O custo em baixas requisições é compensado pela granularidade e escalabilidade adicionados pelo sistema *Serverless*.

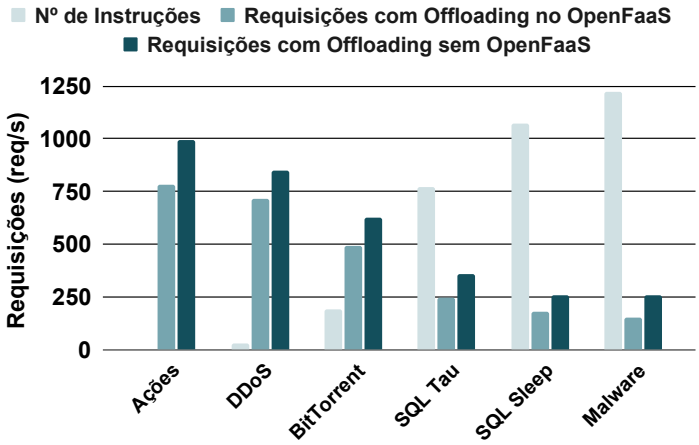

Figura 6. Taxa de requisições.

5. Limitação

Todos os filtros que percorrem e verificam o conteúdo do payload precisam utilizar a diretiva `loop unroll` do compilador `Clang`. Isso acontece devido ao verificador de código eBPF suportar apenas laços limitados. O uso dessa diretiva faz com que o número de instruções cresça de acordo com o tamanho da ER. Dependendo do tamanho da ER do filtro, o número de instruções geradas pode ser maior que o número de instruções suportado pela Netronome, 131.072 instruções. Essa limitação não ocorre em filtros que realizam verificações no cabeçalho do pacote ou utilizam ERs em partes definidas do pacote, por exemplo, o filtro BitTorrent. Nesse filtro, a ER é sempre aplicada no início do pacote, eliminando a necessidade de percorrer o payload e diminuindo o número de instruções. Tecnologias similares ao eBPF, por exemplo, P4 não suportam uso de laços.

6. Trabalhos relacionados

A utilização de arquiteturas *Serverless* no contexto de processamento de pacotes em SmartNICs é um tópico atual e recorrente discutido na literatura da área. Entretanto, nenhum trabalho relacionado da literatura combina programabilidade e flexibilidade do paradigma *Serverless* com *offloading* de filtros de um SDI sobre uma SmartNIC. A tabela 2 apresenta uma comparação do SDI proposto com trabalhos relacionados da área. A comparação foi baseada nos seguintes tópicos: modelo da SmartNIC, suporte ao paradigma *Serverless*, programabilidade, *offloading* em tempo de execução, e comunicação com hospedeiro.

Tabela 2. Comparação trabalhos relacionados.

Trabalho	**SmartNIC**	***Serverless***	**Programabilidade**	***Offloading***	**Hospedeiro**
Este trabalho	Netronome Agilio CX 2x10GbE	Sim	Qualquer função escrita em eBPF/XDP	Sim	Sim
[Hypolite et al. 2020]	Netronome NFP-600	Não	Filtros de rede escritos em P4	Não	Não
[Pacífico et al. 2020]	NetFPGA SUME	Sim	Funções de rede padrão escritas em eBPF	Sim	Não
[Liu et al. 2019b]	Cavium LiquidIO	Sim	Alguns microsserviços	Não	Não
[Choi et al. 2019]	Netronome Agilio CX 2x10GbE	Sim	Funções *lambda* escritas em Micro C	Sim	Sim
[Miano et al. 2019]	Não especificada	Não	Funções de mitigação de ataques DDoS	Sim	Não

Hypolite et al. [Hypolite et al. 2020] propuseram o sistema denominado Deep Match. Deep Match utiliza filtros de segurança da rede com expressões regulares para avaliar o conteúdo de pacotes em busca de ameaças. Os filtros são implementados em P4, uma linguagem de domínio específico limitada se comparado com o eBPF/XDP. Além disso, neste trabalho nenhum mecanismo capaz de adicionar flexibilidade e escalabilidade ao sistema, tal como, o gerenciamento de funções usando um arcabouço Serverless presente em nosso trabalho é suportado.

Pacífico et al. [Pacífico et al. 2020] desenvolveram um sistema de processamento de pacotes *Serverless* com *offloading* de funções C eBPF sobre a SmartNIC NetFPGA SUME. O sistema proposto é limitado ao número de núcleos de processamento (apenas quatro) enquanto a Netronome suporta um número maior de núcleos. Além disso, o sistema não suporta XDP em conjunto com eBPF. Netronome suporta comunicação com o hospedeiro. O SDI proposto pode utilizar essa característica para processamento offline de pacotes no *kernel*. Essa característica não é suportada em Pacífico et al.

Liu et al. [Liu et al. 2019b] apresentaram a plataforma de execução de microsserviços em SmartNICs denominada E3. E3 funciona como um acelerador de microsserviços implementado em hardware para servidores convencionais de *data centers*. Os autores avaliaram o ganho de eficiência energética ao realizar o *offloading*

de microsserviços sobre SmartNICs. A infraestrutura do E3 é restrita ao provedor de nuvem Azure Service Fabric. Os autores não mencionam a linguagem utilizada no desenvolvimento dos microsserviços que rodam na plataforma.

Choi et al. [Choi et al. 2019] projetaram o arcabouço denominado λ-NIC para contornar a baixa eficiência de CPUs em servidores, executando funções *lambda* diretamente na SmartNIC. No λ-NIC as funções *lambda* são construídas em Micro C, uma versão simplificada da linguagem C. No entanto, as funções escritas em eBPF são mais flexíveis e podem realizar um conjunto amplo de operações essenciais na análise e detecção de ameaças tanto no cabeçalho e *payload* do pacote.

Miano et al. [Miano et al. 2019] discutem os benefícios de realizar *offloading* de funções no dispositivo de rede para processamento rápido de pacotes. Como estudo de caso para avaliar a discussão proposta, os autores criaram um sistema de mitigação de ataques DDoS utilizando eBPF/XDP, rodando em uma SmartNIC. O sistema proposto é limitado à operações de análise e busca de ataques DDoS.

7. Conclusão e Trabalhos Futuros

Neste trabalho foi proposto um SDI *Serverless* capaz de realizar análise e filtragem de pacotes trafegados na rede com programabilidade, flexibilidade e escalabilidade. Nosso sistema permite que usuários construam filtros escritos em eBPF e utilizem esses filtros de forma eficiente e simplificada via OpenFaaS. Além disso, o SDI utiliza o alto poder de processamento e eficiência energética da SmartNIC Netronome para acelerar o processamento de pacotes e reduzir custos operacionais independente do filtro descarregado no dispositivo. Filtros eBPF descarregados no hardware são capazes de analisar o tráfego e definir o destino dos pacotes de modo programável baseado nas regras definidas em cada programa, podendo ser alterados em tempo de execução sem gerar sobrecarga no SDI. Os resultados obtidos demonstram que ataques e tentativas de invasão são detectadas em taxa de linha. Como trabalhos futuros pretendemos adicionar um número maior de hardwares para paralelizar o processamento de pacotes e filtros, e adicionar suporte à funções eBPF de processamento mais complexas, como o uso de chamadas de cauda de programas eBPF para melhorar a performance do SDI.

O presente trabalho foi realizado com o apoio das agências de fomento à pesquisa CNPq, CAPES e FAPEMIG. Agradecemos pelo suporte oferecido.

Referências

Arashloo, M. T., Lavrov, A., Ghobadi, M., Rexford, J., Walker, D., and Wentzlaff, D. (2020). Enabling programmable transport protocols in high-speed nics. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*, pages 93–109.

Bertin, G. (2017). XDP in practice: integrating XDP into our DDoS mitigation. In *Proceedings of the Technical Conference on Linux Networking*, page 5.

Choi, S., Shahbaz, M., Prabhakar, B., and Rosenblum, M. (2019). Lambda-nic: Interactive serverless compute on smartnics. In *Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos*, SIGCOMM Posters and Demos '19, pages 151–152, New York, NY, USA. ACM.

Digital, C. (2017). Roubo de dados vai causar um prejuízo acima de R$ 26 trilhões até 2020. https://www.convergenciadigital.com.br/cgi/cgilua.exe/sys/start.htm?UserActiveTemplate=site&UserActiveTemplate=mobile&infoid=45290&sid=18. Acessado em Fevereiro de 2021.

Docker (2021). Docker hub. https://hub.docker.com/. Acessado em: 11/03/2021.

Docker Inc. (2014). Swarm mode overview. https://docs.docker.com/engine/swarm/. Acessado em: 11/03/2021.

Duarte, L. F. S. (2021). *alpine-ebpf*: An alpine-based image for compiling and running ebpf programs. https://hub.docker.com/repository/docker/lucasfsduarte/alpine-ebpf.

eBPF (2014). Extended berkeley packet filter (ebpf). https://ebpf.io/what-is-ebpf. Acessado em 11/03/2021.

Ellis, A. (2016). Openfaas: Serverless functions made simple for docker and kubernates. https://www.openfaas.com/.

Ellis, A. (2017). Openfaas classic templates. https://github.com/openfaas/templates. Acessado em: 11/03/2021.

Fox, G. C., Ishakian, V., Muthusamy, V., and Slominski, A. (2017). Status of serverless computing and function-as-a-service(faas) in industry and research. *CoRR*, abs/1708.08028.

Google (2017). Google Cloud Functions. https://cloud.google.com/functions/. Acessado em Maio de 2019.

Google Inc. (2015). Production-grade container orchestration. https://kubernetes.io/. Acessado em: 11/03/2021.

Hypolite, J., Sonchack, J., Hershkop, S., Dautenhahn, N., DeHon, A., and Smith, J. M. (2020). *DeepMatch: Practical Deep Packet Inspection in the Data Plane Using Network Processors*, page 336–350. Association for Computing Machinery.

Intel (2021). Intel Smart Network Adapter. https://www.intel.com/content/www/us/en/products/network-io/smartnic.html. Acessado em 11/03/2021.

IO Visor (2016). extreme data path (xdp). https://www.iovisor.org/technology/xdp. Acessado em 11/03/2021.

Juniper (2019). Business losses to cybercrime data breaches to exceed $5 trillion by 2024. https://www.juniperresearch.com/press/press-releases/business-losses-cybercrime-data-breaches. Acessado em 12/02/2021.

Liu, M., Cui, T., Schuh, H., Krishnamurthy, A., Peter, S., and Gupta, K. (2019a). Offloading distributed applications onto smartnics using ipipe. SIGCOMM '19, page 318–333, New York, NY, USA. Association for Computing Machinery.

Liu, M., Peter, S., Krishnamurthy, A., and Phothilimthana, P. M. (2019b). E3: Energy-efficient microservices on smartnic-accelerated servers. In *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, pages 363–378.

Miano, S., Doriguzzi-Corin, R., Risso, F., Siracusa, D., and Sommese, R. (2019). Introducing smartnics in server-based data plane processing: The ddos mitigation use case. *IEEE Access*, 7:107161–107170.

Miao, R., Zeng, H., Kim, C., Lee, J., and Yu, M. (2017). Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 15–28.

Microsoft (2017). Azure Functions. `https://azure.microsoft.com/en-us/services/functions/`. Acessado em Maio de 2019.

Moon, Y., Lee, S., Jamshed, M. A., and Park, K. (2020). Acceltcp: Accelerating network applications with stateful TCP offloading. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 77–92, Santa Clara, CA. USENIX Association.

Netronome (2021). Agilio CX SmartNICs. `https://www.netronome.com/products/agilio-cx/`. Acessado em 11/03/2021.

NH (2021). Vazamento expôs dados de 220 milhões de brasileiros. `https://www.jornalnh.com.br/cotidiano/tecnologia/2021/02/01/vazamento-expos-dados-de-220-milhoes-de-brasileiros--saiba-se-seu-cpf-foi-exposto.html`. Acessado em Fevereiro de 2021.

Nvidia (2021). BlueField SmartNIC Ethernet. `https://www.mellanox.com/products/BlueField-SmartNIC-Ethernet`. Acessado em 11/03/2021.

Pacífico, R. D. G., Duarte, L. F. S., Castanho, M. S., Vieira, L. F. M., Nacif, J. A., and Vieira, M. A. M. (2021). Application layer packet classifier in hardware. In *2021 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE.

Pacífico, R., Duarte, L., Castanho, M., Miranda Nacif, J., and Vieira, M. (2020). Sistema de processamento de pacotes serverless. pages 183–196.

Services, A. W. (2017). AWS Lambda. `https://aws.amazon.com/lambda/`. Acessado em Maio de 2019.

Turull, D., Sjödin, P., and Olsson, R. (2016). Pktgen: Measuring performance on high speed networks. *Computer Communications*, 82:39–48.

Vieira, A. G., Pereira, G. H. A., Freire, J. H. F., Duarte, L. F. S., Pacífico, R. D. G., Pantuza, G., Vieira, M. A. M., Vieira, L. F. M., and Nacif, J. A. M. (2020a). Computação Serverless: Conceitos, Aplicações e Desafios. In *Minicursos do XXXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos (SBRC)*, Rio de Janeiro, RJ, Brasil. SBC.

Vieira, M. A. M., Castanho, M. S., Pacífico, R. D. G., Santos, E. R. S., Câmara Júnior, E. P. M., and Vieira, L. F. M. (2019). Processamento Rápido de Pacotes com eBPF e XDP. In *Minicursos do XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos (SBRC)*, Porto Alegre, RS, Brasil. SBC.

Vieira, M. A. M., Castanho, M. S., Pacífico, R. D. G., Santos, E. R. S., Júnior, E. P. M. C., and Vieira, L. F. M. (2020b). Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications. *ACM Comput. Surv.*, 53(1).