UNIVERSIDADE FEDERAL DO PAMPA

Ronaldo Canofre Mariano dos Santos

Rumo à Notificação Seletiva de Informações de Telemetria *In-band* em Planos de Dados Programáveis Utilizando SmartNICs

Alegrete
2022

Ronaldo Canofre Mariano dos Santos

Rumo à Notificação Seletiva de Informações de Telemetria *In-band* em Planos de Dados Programáveis Utilizando SmartNICs

Dissertação de Mestrado apresentada ao Curso de Mestrado Profissional em Engenharia de Software da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Mestre em Engenharia de Software.

Orientador: Prof. Dr. Marcelo Caggiani Luizelli

Alegrete
2022

Ficha catalográfica elaborada automaticamente com os dados fornecidos
pelo(a) autor(a) através do Módulo de Biblioteca do
Sistema GURI (Gestão Unificada de Recursos Institucionais) .

S237r Santos, Ronaldo Canofre Mariano dos
Rumo à Notificação Seletiva de Informações de Telemetria In-band em Planos de Dados Programáveis Utilizando SmartNICs / Ronaldo Canofre Mariano dos Santos.
134 p.

Dissertação(Mestrado)-- Universidade Federal do Pampa, MESTRADO EM ENGENHARIA DE SOFTWARE, 2022.
"Orientação: Marcelo Caggiani Luizelli".

1. Telemetria de Rede In-band. 2. Programabilidade no Plano de Dados. 3. Monitoramento de Rede. I. Título.

RONALDO CANOFRE MARIANO DOS SANTOS

RUMO À NOTIFICAÇÃO SELETIVA DE INFORMAÇÕES DE TELEMETRIA IN-BAND EM PLANOS DE DADOS PROGRAMÁVEIS UTILIZANDO SMARTNICS

Dissertação/Tese apresentada ao Programa de Pós-Graduação em Engenharia de Software da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Mestre em Engenharia de Software.

Dissertação defendida e aprovada em: 15 de julho de 2022.

Banca examinadora:

_____

Prof. Dr. Marcelo Caggiani Luizelli

Orientador

UNIPAMPA

_____

Prof. Dr. Fábio Diniz Rossi

IFFAR

_____

Prof. Dr. Roberto Irajá Tavares da Costa Filho

IFSul

Assinado eletronicamente por **MARCELO CAGGIANI LUIZELLI**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 26/07/2022, às 14:05, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **Roberto Irajá Tavares da Costa Filho**, **Usuário Externo**, em 26/07/2022, às 14:07, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **Fábio Diniz Rossi**, **Usuário Externo**, em 26/07/2022, às 14:08, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

A autenticidade deste documento pode ser conferida no site https://sei.unipampa.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **0876769** e o código CRC **C43DA867**.

RESUMO

Manter a disponibilidade e a qualidade dos serviços é indispensável em qualquer ambiente de rede, sendo o monitoramento uma prática essencial para atingir esses objetivos. O uso de mecanismos de monitoramento proporciona visibilidade sobre o ambiente gerenciado, entregando informações essenciais que permitem aos administradores de rede desempenhar melhor suas tarefas. A telemetria de rede *in-band* é uma abordagem promissora para monitoramento de rede quase em tempo real. Por meio do uso dos pacotes que transitam pela rede, ela possibilita visibilidade ampla e precisa do ambiente monitorado. O uso dos pacotes existentes, permite realizar a coleta e transporte transporte das informações que serão recuperadas em um ponto específico da rede, denominado coletor INT. No entanto, a medida que uma grande quantidade de dados passa a trafegar pela rede, uma análise detalhada e com tempo de resposta eficiente de todas as informações pode afetar severamente o desempenho de aplicativos e serviços em execução. Isso pode ocorrer devido a fatores como: espaço disponível nos pacotes, largura de banda consumida e variação dos quadros transmitidos. Assim, apesar dos esforços recentes para coordenar efetivamente a coleta de informações, uma maneira eficaz de relatar com sabedoria os dados coletados, ainda é uma lacuna presente. Assim sendo, este trabalho tem como objetivo apresentar uma alternativa ao problema de seleção eficiente dos dados a serem encaminhados para um coletor INT, juntamente com a redução de sobrecarga dos dados enviados. Dessa forma, é proposto um mecanismo de relatório seletivo totalmente implementado em um dispositivo programável através da linguagem P4 e rotinas Micro-C. As análises realizadas sobre os fluxos se utilizam de uma média móvel exponencialmente ponderada, calculada dentro do plano de dados, contornando algumas das rigorosas restrições de capacidade de processamento e limitações de uso de memória, com base no que será visto no decorres deste trabalho. Os resultados obtidos demonstram que de uma forma geral, a abordagem proposta apresenta uma melhora considerável em comparação com as técnicas tradicionais de monitoramento, e mesmo que adicione uma sobrecarga na latência, está ainda é inferior aos métodos mais comuns. Isso ao mesmo tempo que apresenta propostas para contornar as limitações observadas.

Palavras-chave: Telemetria de Rede *in-band*, Programabilidade no Plano de Dados, Monitoramento de Rede.

ABSTRACT

Maintaining the availability and quality of services is essential in any network environment, and monitoring is an essential practice to achieve these goals. The use of monitoring mechanisms provides visibility into the managed environment, delivering essential information that allows network administrators to better perform their tasks. *In-band* network telemetry is a promising approach to near real-time network monitoring. Through the use of packets that travel through the network, it provides broad and accurate visibility of the monitored environment. The use of existing packages allows collecting and transporting the information that will be retrieved at a specific point on the network, called the INT collector. However, as a large amount of data passes through the network, a detailed and time-efficient analysis of all the information can severely affect the performance of running applications and services. This can occur due to factors such as: available space in packets, bandwidth consumed and variation of transmitted frames. Thus, despite recent efforts to effectively coordinate the collection of information, an effective way to wisely report the collected data is still a present gap. Therefore, this work aims to present an alternative to the problem of efficient selection of data to be forwarded to an INT collector, along with reducing the overhead of the data sent. Thus, a selective reporting mechanism fully implemented in a programmable device through the P4 language and Micro-C routines is proposed. The analyzes carried out on the flows use an exponentially weighted moving average, calculated within the data plane, circumventing some of the strict restrictions on processing capacity and memory usage limitations, based on what will be seen in the course of this work. The results obtained demonstrate that, in general, the proposed approach presents a considerable improvement compared to traditional monitoring techniques, and even if it adds an overhead in latency, it is still lower than the most common methods. At the same time, it presents proposals to circumvent the observed limitations.

Key-words: In-band Network Telemetry, Data Plane Programmability, Network Monitoring.

LISTA DE FIGURAS

LISTA DE SIGLAS

APF *Analisador preliminar de fluxo*

API *Application Programming Interface*

ASIC *Application-Specific Integrated Circuit*

ATM *Asynchronous Transfer Mode*

BMv2 *Behavioral Model version 2*

CLS *Cluster Local Scratch*

CPP *command-push-pull*

CPU *Central Processing Unit*

CTM *Cluster Target Memory*

DDoS *Distributed Denial of Service*

DDR *Double-Data-Rate*

DoS *Denial of Service*

DP Desvio Padrão

DPDK *Data Plane Development Kit*

DRAM *Dynamic Random Acess Memory*

DUT *Device Under Test*

eBPF *Extended Berkeley Packet Filters*

EMEM *External Memory*

FMC FPGA *Mezzanine Card*

ForCES *Forwarding and Control Element Separationt*

FPC *Flow Processing Cores*

FPGA *Field-Programmable Gate Array*

Gbps Gigabits por segundo

HDL *Hardware Description Languages*

ICMP *Internet Control Message Protocol*

IETF *Internet Engineering Task Force*

IMEM *Internal Memory*

INT *In-band Network Telemetry*

IP *Internet Protocol*

IPv4 *Internet Protocol version 4*

JSON *JavaScript Object Notation*

KB Kilobyte

LPM *Longest Prefix Match*

MAC *Media Acess Controll*

MB Megabyte

Mbit Megabits

MIB *Management Information Base*

MMEP *Média Móvel Expondencial Ponderada*

MP *Média Ponderada*

NAT *Network Address Translation*

NCP *Network Control Point*

NetFPGA *Network Field-Programmable Gate Array*

NIC *Network Interface Card*

NOS *Network Operating System*

ns *nanossegundos*

OAM *Operation Administration and Maintenance*

P4 *Programming Protocol-independent Packet Processors*

PCI *Peripheral Component Interconnect*

PISA *Protocol-Independent Switch Architecture*

PPC *Packet Processing Cores*

PPD Programabilidade no Plano de Dados

pps pacotes por segundo

PSA *Portable Switch Architecture*

RFC *Request for Comments*

RLDRAM *Reduced Latency DRAM*

SDN *Software defined networking*

SFP+ *Samll Form Factor Pluggable Pluss*

SmartNIC *Smart Network Interface Card*

SNMP *Simple Network Management Protocol*

SoC *systen-on-chip*

SRAM *Static Random Acess Memory*

SSL *Secure Socket Layer*

TCP *Transmission Control Protocol*

UDP *User Daclassram Protocol*

VLAN *Virtual Local Area Network*

SUMÁRIO

1 INTRODUÇÃO

1.1 Contexto e Motivação

A manutenção da disponibilidade, da qualidade e das métricas de desempenho de uma infraestrutura de rede é essencial para a correta operação de serviços e aplicações. A utilização de mecanismos de monitoramento permite manter uma visibilidade do ambiente gerenciado, coletando informações da infraestrutura. A análise destas informações permite aos administradores e operadores melhorar a tomada de decisão, tais como *(i)* programar recursos; *(ii)* medir o desempenho dos serviços; *(iii)* planejar a operação (por exemplo, engenharia de tráfego); *(iv)* prever ou rastrear eventos de rede (por exemplo, ataques de rede); e *(v)* identificar falhas, anomalias ou violações de políticas. Dessa forma, a capacidade de identificação e solução de eventuais eventos anômalos nas infraestruturas de rede está limitada à visibilidade que se tem da mesma.

A telemetria *in-band* de rede (INT –*In-band Network Telemetry*) é uma abordagem promissora de monitoramento de rede que permite ampliar a visibilidade do ambiente monitorado (JEYAKUMAR et al., 2014; LIU et al., 2018; PAN et al., 2019). De forma resumida, esta abordagem permite que pacotes de fluxos ativos sejam instruídos a recuperar e encapsular um conjunto de estatísticas de monitoramento de baixo nível diretamente do plano de dados. As informações recuperadas são transportadas em pacotes ao longo de seu caminho de roteamento e, em algum ponto da infraestrutura, extraídas e enviadas a um coletor INT. Posteriormente, as informações do coletor são repassadas para aplicações de monitoramento que processam e reagem aos eventos observados. Alguns exemplos destas estatísticas incluem os metadados existentes nas arquiteturas presentes em dispositivos do plano de dados, tais como o tempo de processamento por pacote ou a utilização de uma determinada fila. Entretanto, pode-se programar os dispositivos para que estes computem/extraiam métricas específicas e apropriadas para uma determinada aplicação de monitoramento. Por exemplo o *inter-packet gap*, que consiste na computação do intervalo de tempo entre pacotes de um mesmo fluxo (JOSHI et al., 2018; SINGH et al., 2020).

A Figura 1 ilustra o procedimento INT em detalhes. Primeiro, um dispositivo INT *source* encapsula e instrui os pacotes dos fluxos de produção para a realização da recuperação de metadados, conforme ilustrado no passo (1). O encapsulamento é realizado utilizando campos de protocolos disponíveis, como por exemplo, o campo opções do IPv4, ou por meio do re-encapsulamento do tráfego por completo, realizado com o encapsulamento INT. Dentro do novo encapsulamento estão as instruções de quais informações devem ser coletadas e em quais dispositivos. Então, os dispositivos intermediários (INT *transit*) processam as instruções adicionadas nos pacotes e adicionam as informações requisitadas de telemetria, conforme ilustrado nos passos (2)–(5). Por fim, um dispositivo final (INT *sink*) realiza o desencapsulamento, extrai as informações de telemetria e as

envia para um coletor INT, ilustrado no passo (6).

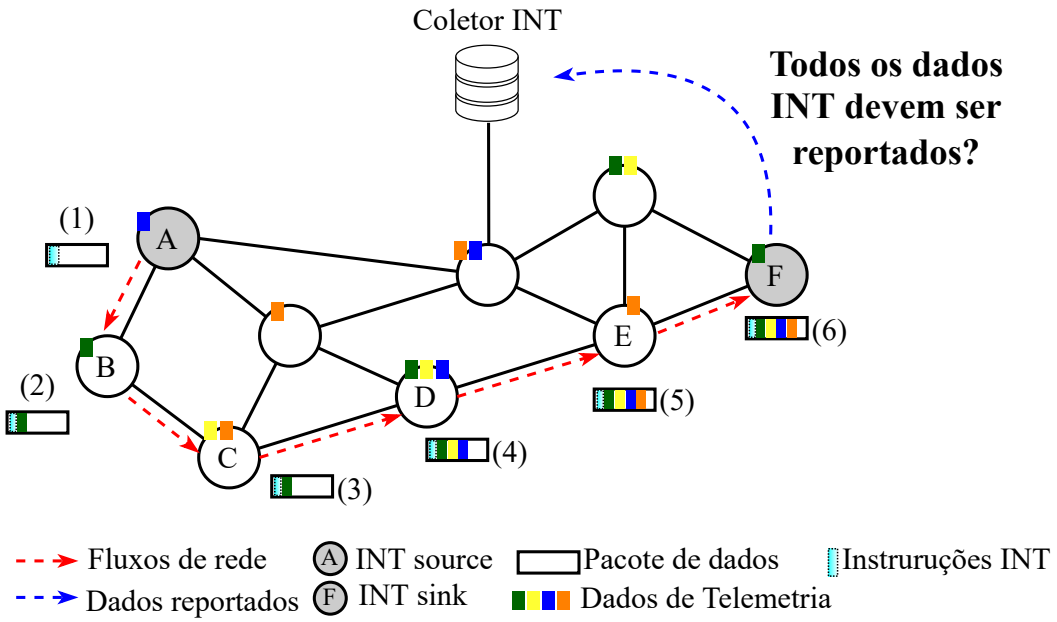


Figura 1 – Visão geral do problema de pesquisa.

Neste exemplo, o pacote de um fluxo f_n é utilizado para recuperar os níveis de ocupação de fila instantâneos de todos os roteadores do caminho que percorre, compreendido de A até F. Dessa forma é possível identificar com maior precisão potenciais eventos anômalos que são dificilmente identificados a partir da coleta de dados agregados utilizando monitoramento tradicional tal como o protocolo SNMP. Um exemplo de problema recorrente em infraestruturas de *data centers* é a ocorrência de *micro burst*. Este problema consiste na inundação temporária e de curta duração (isto é, de alguns nanosegundos) de algumas filas de entrada/saída dos roteadores das infraestruturas, causadas principalmente por rajadas de tráfego de aplicações concorrentes que compartilham os dispositivos. A consequência deste tipo de evento é o congestionamento ou descarte observado em alguns pacotes dos fluxos existentes. Como o evento é transitório e de curta duração, as métricas agregadas coletadas pelo monitoramento tradicional mascaram a existências de tais eventos, inviabilizando a identificação e correção dos eventos.

1.2 Problema

Ao longo dos últimos anos, INT tem sido utilizado por uma quantidade substancial de aplicações incluindo, por exemplo, a identificação de anomalias/intrusões em infraestruturas de rede. A implementação desse mecanismo de monitoramento tem sido viabilizada principalmente pela rápida adoção de dispositivos de rede programáveis e pela popularização de linguagens de domínio-específico como P4 (BOSSHART et al., 2014). Trabalhos recentes nesta área tem concentrado os esforços principalmente na direção da orquestração da coleta de dados de telemetria *in-band* (PAN et al., 2019; HOHEMBER-

GER et al., 2019; SAQUETTI et al., 2021) de modo a aumentar a visibilidade dos eventos ocorridos na infraestruturas.

No entanto, apesar dos esforços de pesquisa existentes neste domínio, pouco ainda foi feito para reportar de maneira eficiente as informações de telemetria para um coletor INT. Como mencionado anteriormente, a Figura 1 ilustra o processo de coleta de informações de telemetria *in-band* pelo pacote de um fluxo f_n. Ao chegar em um dispositivo INT *sink*, os dados são extraídos e reportados ao coletor. Abordagens tradicionais de monitoramento que empregam INT reportam de maneira indiscriminada as informações coletadas. No caso em que todos os dados de telemetria são reportados a um coletor INT, pode haver:

1. *Uso excessivo dos enlaces da infraestrutura entre o INT sink e o coletor INT.* Por exemplo, se considerarmos um enlace de rede de 10 Gbps enviando pacotes de 64 Bytes, teríamos 14,88 milhões de pacotes por segundo. Sendo coletado 1 Byte de informação INT a cada dispositivo ao longo do caminho, o volume de tráfego de rede necessário a ser relatado por segundo seria de 118 Mbit multiplicado pelo comprimento do caminho do fluxo. Esse volume de dados relatado pode aumentar substancialmente se assumirmos a arquitetura de referência para dispositivos programáveis[1] onde cada dispositivo possui pelo menos 30 Bytes de metadados;

2. *Degradação do desempenho de processamento de pacotes dos dispositivos que implementam o INT sink.* Para enviar o pacote de rede para o coletor INT, os dispositivos programáveis que implementam o *INT sink* precisam utilizar primitivas de recirculação/clonagem de pacotes para duplicar o pacote – o que reduz drasticamente o desempenho em termos de taxa de transferência e latência (VIEGAS et al., 2021a); e

3. *Sobrecarga das aplicações que implementam o coletor INT.* No geral, tais aplicações são responsáveis por processar os dados brutos de telemetria coletados na infraestrutura e pela tomada de decisão. O processamento de pacotes de maneira escalável em nível de aplicação ainda é um desafio. Para se processar altas taxas de pacotes (por exemplo, maiores que 10 Gbps) em nível de aplicação é necessário utilizar tecnologias como Intel DPDK[2].

1.3 Objetivos e Contribuições

Neste trabalho, propõe-se um mecanismo seletivo para reportar informações de telemetria *in-band* para coletores INT, desenvolvido inteiramente sobre SmartNICs. O mesmo baseia-se no cômputo de médias móveis exponencialmente ponderadas diretamente no plano de dados, fazendo uso de informações históricas que auxiliam na tomada de

[1] <https://github.com/p4lang/p4c/blob/main/p4include/v1model.p4>
[2] https://www.intel.com.br/content/www/br/pt/communications/data-plane-development-kit.html

decisão – isto é, se é necessário reportar um dado ao coletor INT. Para tanto, assume-se que o INT *sink* é implementado por meio de uma interface programável, e portanto, a decisão de reportar ou não os dados de telemetria é inteiramente realizado pelo plano de dados. Considera-se também, que as interfaces utilizadas, conforme relatado pela literatura (VIEGAS et al., 2021a), possuem limitações de uso de memória e restrições em termos de recursos de processamento, como por exemplo, falta de operações de ponto flutuante.

Desta forma, as principais contribuições deste trabalho se desdobram em:

1. projetar, implementar e avaliar um mecanismo seletivo, que permita reportar de forma eficiente, as informações de telemetria *in-band* necessárias, inteiramente no plano de dados, de forma adequada às restrições arquiteturais das SmartNICs;

2. realizar uma verificação das limitações existentes em dispositivos de rede programáveis para se projetar e implementar operações mais complexas, realizando a apresentação de propostas para contorná-las;

3. realizar o registro de *software* e a posterior divulgação para comunidade, possibilitando a reprodutibilidade dos experimentos conduzidos.

1.4 Organização

O restante desde trabalho está organizado da seguinte maneira:

- **Capítulo 2**: revisão de literatura acerca dos conceitos de programabilidade de rede de forma geral e das possíveis formas de realização do monitoramento de rede, abordando por fim os dispositivos de rede programáveis.

- **Capítulo 3**: abordagem dos trabalhos relacionados a proposta apresentada, juntamente com uma avaliação comparativa dos trabalhos encontrados e relação aos objetivos almejados com esta proposta.

- **Capítulo 4**: detalhamento do funcionamento e implementação, bem como os detalhes e limitações da arquitetura e linguagem e os impactos sobre o desenvolvimento da aplicação.

- **Capítulo 5**: apresentação do ambiente de trabalho, do caso de uso e metodologia utilizada para testes e análise e dos resultados obtidos, bem como uma avaliação dos mesmos.

- **Capítulo 6**: explanação sobre as principais contribuições deste trabalho, apresentando das considerações finais e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, apresenta-se inicialmente uma revisão da literatura sobre os principais conceitos relacionados com programabilidade de infraestrutura de redes. Após, abordam-se os conceitos relacionados com o monitoramento de infraestruturas de rede tradicionais e programáveis. Por fim, discute-se as arquiteturas existentes de interfaces de redes programáveis.

2.1 Programabilidade de Infraestruturas de Rede

As redes de computadores são ambientes complexos, de difícil gerenciamento, e compostas por uma ampla variedade de dispositivos tais como *firewalls*, NAT (*Network Address Translation*), balanceadores de carga e, principalmente, por dispositivos de encaminhamento. Estes dispositivos representam a maior porção dos equipamentos que compõe uma infraestrutura rede, sendo responsáveis pelo maior custo operacional envolvido, principalmente, devido a variedade de interfaces de gerenciamento dos diversos fabricante (FEAMSTER; REXFORD; ZEGURA, 2014).

A programabilidade de rede surgiu, inicialmente, como uma alternativa à extensa variedade de interfaces de gerenciamento existente nos dispositivos de encaminhamento. O objetivo consiste em tornar estes equipamentos de rede programáveis, similarmente ao que já ocorre na programação de CPUs. Ao permitir a programação de tais dispositivos, possibilita-se a criação de funcionalidades personalizadas à cada infraestrutura ou domínio de aplicação, tais como interfaces de gerenciamento unificadas. Assim, o gerenciamento e a operação destes ambientes torna-se menos dependente dos fabricantes de dispositivos. Isso devido a adição de um certo nível de programabilidade, ao invés de funcionalidades atreladas à circuitos integrados ASIC (*Application-Specific Integrated Circuit*). Consequentemente, aumenta-se a flexibilidade em relação à adição/remoção de funcionalidades e ao controle sobre os dispositivos (FEFERMAN et al., 2018).

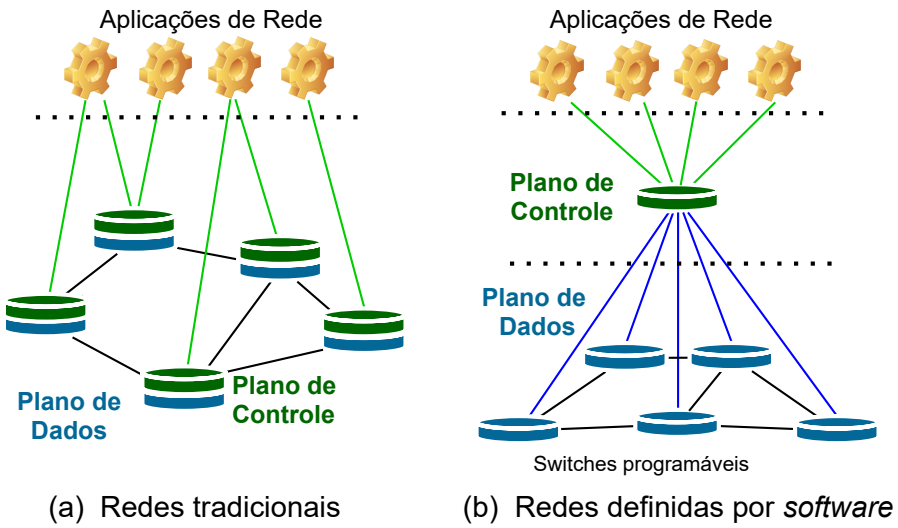
No entanto, embora tenha ganhado destaque nos últimos anos, a programabilidade de infraestruturas de rede já vem sendo abordada em algum nível, em trabalhos realizados nas últimas três décadas, tal como as redes ativas e redes ATM programáveis. De forma similar, houveram também iniciativas para a separação dos planos de controle e de dados em redes NCP (*Network Control Point*[1]), Tempest (virtualização de redes ATM) e ForCES (*Forwarding and Control Element Separationt*). Tais esforços de pesquisa serviram como base para a arquitetura de Redes Definidas por *Software* (ZANETTI, 2020).

[1] Técnica empregada em serviços de telefonia *toll-free* (como 0800), que se utiliza de geolocalização para sua concessão

2.1.1 Redes Definidas por Software

As Redes Definidas por *Software* (SDN – *Software defined networking*), possibilitam basicamente que aplicações em *software* realizem a programação de dispositivos de rede de forma dinâmica, controlando seu comportamento de forma centralizada e inteligente. Tais características são obtidas com a utilização de um conjunto de técnicas que viabilizam o desenvolvimento de serviços de rede de forma determinística, dinâmica e escalável (HALEPLIDIS et al., 2015). O termo SDN foi inicialmente definido como uma arquitetura que realiza o gerenciamento do plano de dados a partir de um plano de controle desvinculado e remoto. Essa separação dos planos de dados e de controle teve origem em trabalhos de pesquisa da Universidade de Stanford, os quais resultaram na criação da interface de comunicação OpenFlow (MCKEOWN et al., 2008). No entanto, recentemente o termo SDN tem sido utilizado para se referir ao uso de programabilidade em qualquer nível da infraestrutura de rede.

Os dispositivos de rede tradicionais são funcionalmente divididos em plano de controle e de dados. No entanto, embora logicamente independentes, eles são verticalmente integrados, ou seja, implementados de forma monolítica sobre um mesmo dispositivo físico conforme ilustrado pela Figura 2(a). Nesta forma de operação, existem algumas desvantagens, tais como o gerenciamento descentralizado e a complexidade de configuração de uma variedade de dispositivos distintos realizada de forma individual. A arquitetura SDN possibilita implementar a programabilidade de rede mediante da separação efetiva desses planos. O plano de dados é mantido nos dispositivos físicos e move-se o plano de controle para uma unidade de processamento externa, um servidor por exemplo, permitindo o gerenciamento de forma remota, centralizada e por meio de interfaces unificadas. A Figura 2(b) ilustra o conceito de SDN.



(a) Redes tradicionais (b) Redes definidas por *software*

Figura 2 – Arquitetura Tradicional (a) e SDN (b)

Na arquitetura SDN, o plano de controle é a entidade inteligente da infraestrutura,

sendo implementado na forma de um sistema operacional de rede (NOS – *Network Operating System*). Este sistema operacional é implementado seguindo um modelo cliente/servidor, sendo responsável pelo fornecimento das abstrações e funcionalidades necessárias para viabilizar a programabilidade desejada. O plano de controle se responsabiliza por toda a lógica da rede, definindo e atualizando por exemplo, as tabelas de roteamento e políticas de rede, determinando como os pacotes serão processados (HALEPLIDIS et al., 2015). Essas definições são então aplicadas no plano de dados, de um ou mais nós da infraestrutura.

O plano de dados é composto pelos nós da rede que são responsáveis por realizar o encaminhamento de pacotes de forma rápida e eficaz. Esse processamento é realizado por meio de um *pipeline*[2], em que as informações dos pacotes são analisadas e o seu processado é realizado de acordo com as definições recebidas do plano de controle. Outras funções do plano de dados incluem o repasse de informações sobre os datagramas e a coleta e envio de estatísticas sobre fluxos da rede para o controlador (CAMERA, 2020). Como exemplos de fluxos de rede, pode-se utilizar um conjunto de pacotes que estabelecem conexão em uma mesma porta ou pertencem a uma determinada faixa de rede, ou seja, possuem uma característica comum.

2.1.1.1 Interfaces de comunicação

A troca de informações que ocorre na arquitetura SDN é realizada por *Application Programming Interfaces* (APIs), as quais disponibilizam uma abstração para que os administradores e/ou aplicações de rede realizem a programação dos dispositivos da infraestrutura (FEAMSTER; REXFORD; ZEGURA, 2014). A comunicação entre as aplicações de rede e o plano de controle adota a API *Northbound Interface*, enquanto que a comunicação entre o controlador e o planos de dados faz uso da API *Southbound Interface*.

De forma geral, *Northbound Interface* é o nome dado a qualquer interface que se comunique com uma camada superior. No contexto de SDN, é por meio desta interface que a troca de informações entre o controlador e as aplicações ocorre, permitindo que as configurações a serem aplicadas sejam repassadas ao NOS (NASCIMENTO, 2018). Analogamente, *Southbound Interface* se refere a uma interface que se comunique com a camada inferior, sendo utilizada pelo plano de controle para comunicar com o plano de dados, realizando a troca de informações e atualização de configurações. A Figura 3 ilustra o relacionamento entre as interfaces.

Atualmente não existe uma padronização para a API *Northbound*, fazendo com que grande parte dos controladores (por exemplo: Floodlight, Trema, NOX) definam suas próprias APIs para viabilizar a comunicação, tais como as APIs REST, REST-CONF e Onix. Já para a interface *Southbound*, a API mais utilizada é o OpenFlow (MCKEOWN et

[2] Um fluxo de instruções segmentadas em etapas, percorridas pelos pacotes ao passarem pelo equipamento, abordado de forma mais aprofundada na seção 2.1.2.

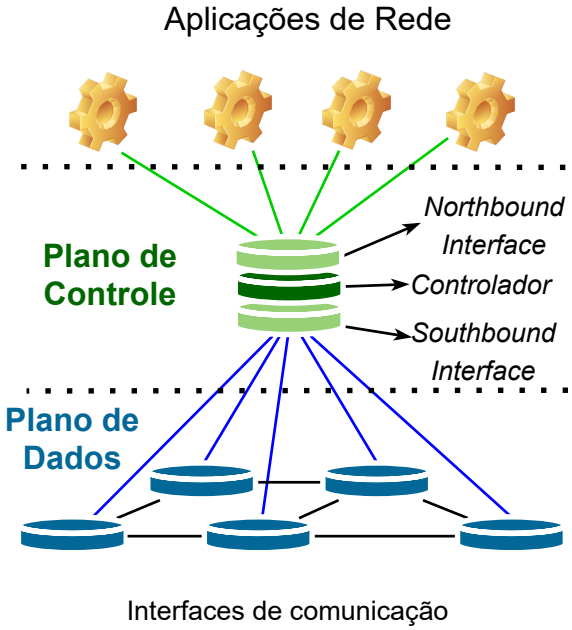


Figura 3 – Interfaces de comunicação do plano de controle

al., 2008), baseada no protocolo OpenFlow, o qual é amplamente adotado e implementado nos dispositivos de rede e suportado amplamente pelos controladores SDN.

2.1.1.2 Protocolo OpenFlow

O protocolo OpenFlow (MCKEOWN et al., 2008) foi desenvolvido como uma forma de execução de protocolos experimentais nas infraestruturas de redes. O objetivo era incentivar a sua inclusão nos dispositivos fornecidos pelos maiores fabricantes, tais como Cisco, HP, Juniper, Extreme, Arista. Mediante da definição de uma abstração para a programação do plano de dados de dispositivos de rede, este protocolo permite que as tabelas de fluxo sejam programadas para, por exemplo, determinar o encaminhamento dos pacotes. Ainda, permite que modificações sejam aplicadas aos cabeçalhos dos pacotes que são encaminhados, possibilitando por exemplo, realizar a troca do identificador da VLAN[3] ou o endereço MACs.

A atualização destas tabelas é realizada por meio de troca de mensagens entre o controlador e o dispositivo de encaminhamento, com a utilização de um canal seguro com criptografia SSL (*Secure Socket Layer*[4]). As mensagens OpenFlow seguem um padrão definido pela interface de modo a permitir que instruções e campos de cabeçalho sejam programadas pelo controlador SDN. Essas mensagens podem ser simétricas, assíncronas ou iniciadas pelo controlador. Mensagens simétricas podem ser utilizadas para comunicar

[3] Rede logicamente independente configurada sobre um dispositivo físico, que permite segmentação lógica de uma rede física.

[4] Protocolo de segurança que cria um link criptografado entre cliente e servidor, garantindo a autenticidade e veracidade dos dados

uma nova conexão (*hello*) ou identificar a disponibilidade de conexão (*echo vendor*). Mensagens assíncronas são realizadas pelo dispositivo de encaminhamento para indicar eventos na rede (*error*) ou mudança de estado (*packet-in, flow-removed, port-status*). Por fim, as mensagens geradas pelo controlador gerenciam e inspecionam o estados dos dispositivos (*features, configuration, modify-state, send-packet, barrier*).

Em um ambiente já configurado, quando um pacote chega a um dispositivo de encaminhamento, os cabeçalhos dos pacotes são comparados aos valores existentes nas tabelas de fluxos e com base nessa comparação, são tomadas as ações correspondentes. Quando não existe correspondência, o pacote ou somente seu cabeçalho é encaminhando ao NOS, para que seja então definido o tratamento a fluxo correspondente a este pacote. Após os dispositivos de encaminhamento são então atualizados de acordo com as mensagens enviadas pelo controlador. Como exemplos de ações associadas aos fluxos tem-se a modificação de campos de cabeçalho, o descarte dos pacotes e encaminhamento dos pacotes para determinada porta ou diretamente para o controlador.

Embora o protocolo OpenFlow tenha se tornado um padrão amplamente difundido com uma constante evolução nos protocolos e cabeçalhos suportados, ainda apresenta ainda algumas limitações. Por exemplo, embora a quantidade de cabeçalho tenha aumentado (de 12 campos na versão 1.0 para 44 campos na versão 1.5.1), a flexibilidade necessária para inclusão de novos campos ou ações ainda não é oferecida (CAMERA, 2020). Cabe ainda ressaltar que a própria evolução constante do protocolo se apresenta como um problema, uma vez que muitos fabricantes ainda implementam e suportam somente as versões mais antigas do protocolo (SONG, 2013).

Outras limitações podem ser vistas com relação ao plano de dados, pois o mesmo necessita entender o formato dos pacotes trocados para que seja possível realizar algum tipo de programação. Adicionalmente, do ponto de vista de análise dos fluxos, o plano de dados não armazena informações de estado, não sendo possível realizar um monitoramento ou mudança de comportamento a partir dele, de forma independente do controlador (SONG, 2013). Assim, a programabilidade de rede acaba se concentrando no plano de controle e subutilizando o poder de processamento disponível no plano de dados. A Programabilidade no Plano de Dados (PPD), que será abordada na Seção 2.1.2, surge como uma alternativa para tentar contornar estas limitações e viabilizar uma rede totalmente programável.

2.1.2 Programabilidade no Plano de Dados

A ideia de tornar o plano de dados programável foi apresentada em 2014 por Bosshart et al. (2014), como uma alternativa as constantes atualizações realizadas no protocolo OpenFlow. Como forma de viabilizar essa alternativa de maneira mais descomplicada e independente de protocolo e limitação de cabeçalhos pré-definidos, foi proposta também uma linguagem de domínio específico denominada P4 - *Programming Protocol-independent*

Packet Processors, cuja primeira versão foi lançada em 2014 (p4-14) e atualizada em 2016 para a versão atual, p4-16.

2.1.2.1 Linguagem P4

P4 é uma linguagem de domínio específico para programação do plano de dados em dispositivos de rede, tais como dispositivos de encaminhamento, roteadores, interfaces de rede programáveis, entre outros. A sua adoção permite ao plano de dados, interagir no processamento dos fluxos, tendo sua especificação sob responsabilidade da P4 Language Consortium (CONSORTIUN, 2021). A adoção desta linguagem permite por exemplo: definir quais os protocolos serão suportados, realizar a criação de novos cabeçalhos, analisar dados dos pacotes e fluxos de dados, armazenar e analisar informações, dentre outras personalizações que não se mostravam viáveis no protocolo OpenFlow e em aplicações de rede residindo no controlador.

Segundo (BOSSHART et al., 2014), o principal desafio da implementação de uma linguagem de alto nível consiste em encontrar a expressividade e facilidade de implementação para ampla variedade de soluções de *hardware* e *software* disponíveis. Dessa forma, a linguagem P4 possui três objetivos principais:

- **Independência de protocolo**: os dispositivos devem ser capazes de analisar qualquer tipo de protocolo que seja declarado na sua programação, não sendo vinculados a protocolos de forma fixa.
- **Independência de arquitetura**: o desenvolvimento das aplicações não deve levar em consideração a arquitetura de *hardware* sobre a qual serão executadas.
- **Reconfiguração em campo**: mesmo depois de implementado, deve ser possível alterar a forma como os dispositivos realizam o processamento dos pacotes.

A linguagem P4 é baseada na arquitetura PISA (*Protocol-Independent Switch Architecture*), ilustrada na Figura 4, a qual especifica o modelo de encaminhamento dos pacotes realizado pelos dispositivos. Essa arquitetura é composta por dois componentes básicos: o primeiro, um *parser* programável que identifica os cabeçalhos do pacote e extrai as informações para estruturas locais; o segundo, tabelas de *match-action*, um *pipeline* de ação de correspondência que define o tratamento dado ao pacote de acordo com a correlação dos valores de campos presentes no cabeçalho e na(s) tabela(s). Por fim, após as atualizações necessárias, os pacotes são então remontados ao final da execução do *pipeline*.

Baseado nessa arquitetura, um programa P4 é estruturado em blocos, tais como: *headers* e metadados, *parser* programável, *ingress control*, *egress control* e *deparser*. Nos *headers* e metadados são realizadas as declarações dos pacotes, tipos de dados e estruturas que serão suportadas pelo plano de dados, sendo possível realizar a declaração de novas

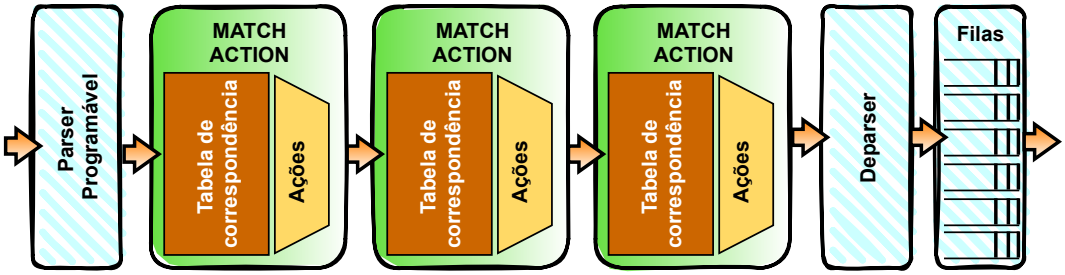


Figura 4 – Visão geral da arquitetura PISA

estruturas ou protocolos. O *parser* é uma máquina de estados finitos – exemplificada na Figura 5 – responsável por detectar em um fluxo de *bits*, os cabeçalhos existentes no pacote, identificando os protocolos implementados. Seus estados são definidos segundo a programação realizada, podendo executar repetições em um mesmo cabeçalho e ignorar cabeçalhos desconhecidos pelo plano de dados.

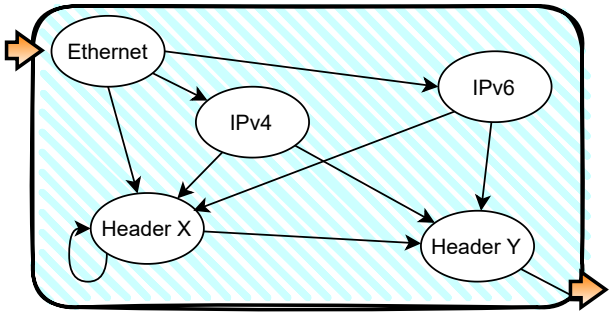


Figura 5 – Comportamento do *parser*

Os blocos de controle *ingress* e *egress* realizam o processamento dos pacotes com base nas informações extraídas no *parser* ou de métricas disponibilizadas pelo dispositivo. Essa ação é realizada por meio de *actions* (funções), que podem ser executadas por tabelas de *match-action* ou diretamente no corpo do bloco de controle. As tabelas de *match-action* por sua vez, executam as ações de acordo com a correspondência de valores entre as informações extraídas dos cabeçalhos e os presentes nas tabelas, funcionando de forma similar a um controle de fluxo realizado com uma estrutura *switch-case*. Em ambos os blocos de controle podem ser recuperadas, removidas, adicionadas ou alteradas informações. No entanto, existem algumas recuperações ou alterações que somente podem ser realizadas em um ou outro bloco, tal como a porta de saída no *ingress* e o registro de data/hora (*timestamp*) que o pacote entra no *pipeline* de saída, no *egress*.

Por fim, o *deparser* realiza o processo inverso ao realizado no *parser*, inserindo os dados novamente nos cabeçalhos e remontando o pacote. Existem ainda os blocos *Verify Checksum* e *Compute Checksum* que realizam respectivamente a verificação do *checksum* do pacote ao entrar no dispositivo e a atualização deste mesmo valor antes que ele seja remontado e enviado para o seu destino. A Figura 6 exemplifica o *pipeline* de execução de um programa P4 com o modelo de encaminhamento abstrato por ele utilizado.

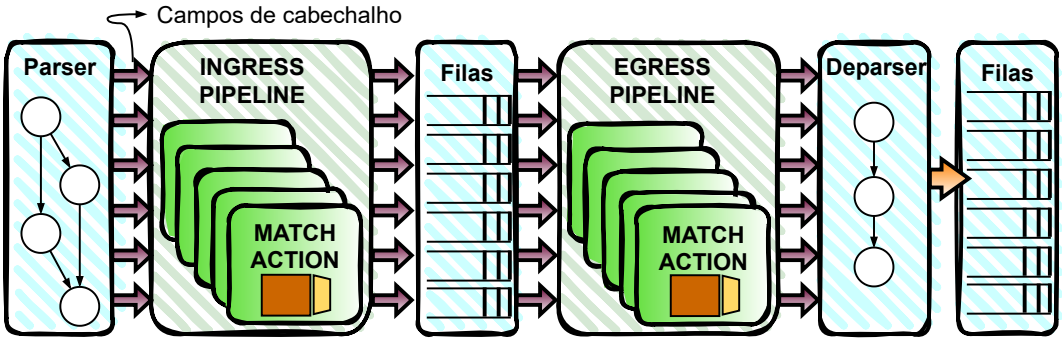


Figura 6 – *Pipeline* de execução do P4

2.1.2.2 Compilador P4

A independência de arquitetura almejada pela linguagem P4 é obtida devido ao seu compilador, responsável por tratar as peculiaridades da plataforma para a qual o programa está sendo compilado. Dessa forma, para cada dispositivo distinto, seja ele físico ou virtual, será necessário um compilador específico. Basicamente o funcionamento de um compilador para linguagem P4 consiste em gerar um formato intermediário único e posteriormente realizar a transformação desse formato para um código de baixo nível suportado pela plataforma onde será executado.

O P4C (CONSORTIUN, 2022) é o compilador de referência para linguagem P4, desenvolvido de forma modular, formado de um *frontend* e um *midend* padrões, os quais podem ser combinados com *backends* específicos. Dessa forma é possível obter um compilador completo, mantendo a padronização da linguagem ao mesmo tempo que se facilita a adoção de *backends* específicos para cada plataforma. Porém, cada fabricante pode realizar modificações em qualquer um dos módulos, para obter um compilador mais personalizado, atendendo assim as especificidades de seus dispositivos (CAMERA, 2020).

Sendo o compilador de referência, o P4C disponibiliza 6 *backends* como modelo:

- **p4c-bm2-ss**: utilizado em ambientes que adotam o modelo comportamental BMv2[5] (*Behavioral Model version 2*). Recebe como entrada um arquivo no formato JSON[6] oriundo do compilador P4 e o interpreta para implementar o comportamento de processamento de pacotes especificado por esse programa.

- **p4c-dpdk**: utilizado para configuração do *pipeline* de dispositivos de encaminhamento em *software* do DPDK, denominado SWX, é compatível somente com a versão P4-16 em arquiteturas PSA (*Portable Switch Architecture* e PSA (*Portable Switch Architecture*.

[5] Modelo comportamental de um dispositivo de encaminhamento em *software*, baseado na arquitetura PISA

[6] *JavaScript Object Notation*, um formato leve de troca de informações/dados entre sistemas.

- **p4c-ebpf**: compatível atualmente com a versão P4-16, é utilizado para obtenção de um código em linguagem C. Posteriormente este código pode ser compilado para eBPF (*Extended Berkeley Packet Filters*) e então carregado no núcleo do sistema operacional Linux para filtragem de pacotes.
- **p4-ubpf**: implementa os mesmos conceitos do p4c-ebpf, no qual é baseado, gerando como resultado um código BPF compatível com a implementação no espaço do usuário.
- **p4-graphs**: utilizado para geração de representações visuais de um programa P4, suportando atualmente apenas a geração de gráficos de fluxos de controle de nível superior.
- **p4test**: um tradutor entre códigos fonte P4 utilizado para realização de testes, depuração e aprendizado interno do compilador.

2.1.2.3 Interface P4 Runtime

As interfaces de comunicação utilizadas anteriormente a PPD, tal como OpenFlow, foram projetadas tendo como base planos de dados estáticos, ou seja, com um conjunto de ações pré-definidas. Ao tornar os planos de dados programáveis, o conjunto de ações existentes tornou-se dinâmico, permitindo um controle limitado por parte das interfaces existentes. Nestes contextos foi apresentada a interface P4 Runtime, como uma nova forma de comunicação entre os controladores SDN e dispositivos programáveis (MARCUZZO et al., 2020).

O protocolo OpenFlow foi projetado para um conjunto específico de casos de uso comum, apresentando uma atualização complexa e difícil em relação a quantidade de protocolos, aplicativos e funcionalidade. Devido a forma como foi projetado, suas atualizações acabam sendo também difíceis de gerenciar. O OpenFlow ainda trabalha realizando somente a ordenação dos campos de um pacote, sendo restrito com relação as ações que devem ser realizadas. O P4 Runtime é uma interface aberta que permite controlar quaisquer dispositivos de encaminhamento cujo comportamento tenha sido especificado na linguagem P4. Permite ainda uma adoção tanto por planos de dados estáticos como dinâmicos, e tendo sido projetado para facilitar a adição de recursos, se torna também facilmente extensível (MCKEOWN, 2017).

Sua utilização pode ser realizada tanto para planos de controle remotos – como as redes SDN, como para planos de dados locais – como nos dispositivos utilizados em redes tradicionais. Em um plano de controle remoto o *pipeline* de encaminhamento é definido pelo programa P4 e o compilador P4 gera o esquema necessário para que a interface P4 Runtime possa adicionar e excluir entradas na tabela de encaminhamento em tempo de execução. Alternativamente, um plano de controle local pode usar o P4 Runtime como uma interface para controlar um plano de dados local no dispositivo de encaminhamento.

2.2 Monitoramento de Rede

O monitoramento de rede consiste na realização de medições que buscam auxiliar na obtenção da visão geral da rede, mostrando seu comportamento atual. Por meio dele é possível obter informações do estado recente dos equipamentos ou da rede, tais como CPU (*Central Processing Unit*), memória, tráfego nas interfaces, consumo de banda e tempo de respostas (CAMERA, 2020).

Segundo a RFC 7799 (MORTON, 2016), estas medições podem ser realizadas de acordo com três métodos: ativo, passivo e híbrido. Os métodos de monitoramento ativo injetam fluxos de pacotes que incluem campos ou valores destinados a medição, sendo geralmente conhecidas a sua origem e destino. No entanto, ao adicionar tráfego à rede para realizar uma medição, estes métodos acabam influenciando os resultados em algum grau. Assim, é necessário que os responsáveis pela execução tratem esse problema de modo a quantificar e/ou minimizar estes efeitos.

Os métodos passivos são realizados com base unicamente em observações dos fluxo de pacotes de interesse, sem manipulação de datagramas, campos ou valores. A execução deste método é realizada em nós específicos distribuídos pela rede e depende da existência de um ou mais fluxos e da ocorrência destes nos pontos de medição determinados. Ainda, a coleta pode ser realizada sobre todos os pacotes que passam pelos pontos definidos ou somente sobre pacotes que correspondam a critérios pré-determinados. Por fim, os métodos híbridos correspondem ao uso de uma combinação dos métodos anteriores, resultando em formas de medição classificadas de acordo com o conjunto de combinações de coleta utilizadas.

Independente do método utilizado, a sua adoção é extremamente útil para o monitoramento do tráfego da rede e coleta de dados estatísticos. No entanto é importante realizar a definição do método a ser empregado, levando em consideração as vantagens e desvantagens, bem como o impacto e os resultados obtidos. Ainda, de acordo com a forma de medição adotada, um coletor independente também pode ser utilizado. Este consiste em um dispositivo presente na rede para captura dos pacotes, e o envio para um monitor externo, que será responsável por processar e/ou exibir as informações. Alguns protocolos amplamente utilizados para esta tarefa são o NetFlow, sFlow e OpenFlow, sendo o primeiro um padrão proprietário e os demais padrões abertos (SILVA et al., 2016).

As formas de medição podem ainda ser classificadas de acordo com a evolução que vem ocorrendo nos últimos anos, sendo divididas em: *(i)* medição tradicional desenvolvida desde 1995; *(ii)* medição baseada em *software*, desenvolvida juntamente com as redes definidas por *Software*; *(iii)* e a telemetria em rede (*network telemetry*), cujo surgimento ocorre juntamente com a ascensão da programabilidade no plano de dados (PPD), desde 2015 (TAN et al., 2021).

2.2.1 Monitoramento Tradicional

A medição tradicional é realizada com o uso de ferramentas de monitoramento, desenvolvidas com base em protocolos que padronizam a comunicação entre os dispositivos, tais como SNMP (*Simple Network Management Protocol*) e ICMP (*Internet Control Message Protocol*). O Protocolo SNMP é o mais difundido e possui três versões disponíveis e em funcionamento. Este protocolo é composto por três componentes: (1) o agente, que implementa o suporte ao protocolo pelo próprio dispositivo a ser monitorado, (2) a MIB (*Management Information Base*) que é a base padrão a partir da qual serão obtidas as informações e (3) o gerente, que corresponde ao sistema que irá centralizar e apresentar as informações coletadas (CASE et al., 1990).

O protocolo ICMP, definido pela RFC 792 (POSTEL, 1981) é utilizado para comunicar informações da camada de rede, sendo o uso mais comum para fornecer relatórios de erros à fonte original. Ele se encontra presente em todos os dispositivos de rede com suporte a pilha de protocolos TCP/IP (*Transmission Control Protocol/Internet Protocol*), sendo disponibilizado de forma integrada ao protocolo IP. Comandos baseados em ICMP, como *ping*[7] e *traceroute*[8], permitem realizar um acompanhamento da rede de forma isolada em um monitoramento reativo ou então em conjunto com outros protocolos para que seja possível obter resultados mais abrangente no controle da rede.

Melhores resultados também podem ser obtidos com ferramentas específicas para o monitoramento de rede, resultantes da associação de comandos e protocolos em um único ambiente. Aplicações como Zabbix (COMPANI, 2022), Nagios (ENTERPRISES, 2022) e Cacti (GROUP, 2021) são exemplos que utilizam os protocolos SNMP e ICMP para fornecer informações mais detalhadas sobre um ambiente, permitindo realizar um acompanhamento mais amplo das informações de cada elemento da rede. Informações de desempenho de hardware, consumo de banda, alterações de latência, espaço em disco, alterações em arquivos e muitas outras, podem ser obtidas de dispositivos de rede, servidores, aplicações e quaisquer outros elementos que estejam conectados (BELENTANI; MARCELLO; FLORIAN, 2018).

No entanto, esta forma de monitoramento baseada em protocolos apresenta algumas deficiências, como por exemplo a limitações na personalização das suas funcionalidades e métodos adotados. Ademais, a dificuldade de detecção de eventos de curta duração, devido ao elevado intervalo de tempo demandando para coleta das métricas, bem como as limitações em ações de configuração e alteração de rede acabam tornando o SNMP inadequado para o uso em tecnologias emergentes (MELO, 2020).

[7] <https://linux.die.net/man/8/ping>
[8] <https://datatracker.ietf.org/doc/html/rfc1393>

2.2.2 Monitoramento SDN

A abertura dos planos de controle e a possibilidade de personalizar o plano de dados, advindos do surgimento das redes SDN e da PPD, impulsionaram também a evolução das formas tradicionais de medição de rede. A programação de estruturas de redes viabilizadas por estes paradigmas proporciona o desenvolvimento de métodos de monitoramento mais personalizados, resultando na coleta de dados mais específicos em um fluxo de dados (CAMERA, 2020). Complementarmente, se mostram também como alternativa a forte dependência de ações do administrador, presente no monitoramento tradicional.

Segundo Tan et al. (2021), desde o surgimento do paradigma SDN, inúmeros trabalhos foram apresentados tanto pela academia como pela indústria, permitindo avaliações de largura de banda disponível (MEGYESI et al., 2017), perda de pacotes (ZHANG et al., 2017), *throughput* (QUEIROZ; CAPRETZ; DANTAS, 2019), latência (YU et al., 2015), rastreamento de caminho (WANG et al., 2017), localização de falha (ZHAO; ZHANG; JIN, 2016), entre outros. Em Rezende (2016) são relacionadas algumas das propostas já apresentadas para monitoramento SDN, tais como OpenNetMon (ADRICHEM; DOERR; KUIPERS, 2014) para monitoramento de largura de banda, OpenSample (SUH et al., 2014) para detecção de *heavy hitters*, e PayLess (CHOWDHURY et al., 2014) para um monitoramento externo com consulta ao controlador.

A PPD permite a construção de novas funções e novos protocolos voltados para monitoramento SDN, resultando em medições de rede mais diretas e eficazes (TAN et al., 2021). Trabalhos recentes nesta área tem apresentado soluções para variados problemas, tais como coleta de informações (LIN et al., 2021; FENG et al., 2020; REZENDE, 2016), balanceamento de carga (COSTA, 2016), gerenciamento de filas (CUGINI et al., 2019; FENG et al., 2019), detecção de ataques de negação de serviço (LAPOLLI; MARQUES; GASPARY, 2019; DIMOLIANIS; PAVLIDIS; MAGLARIS, 2020) incluindo também a telemetria seletiva (KIM; SUH; PACK, 2018; CHOWDHURY; BOUTABA; FRANÇOIS, 2021).

No entanto, embora os métodos e meios de monitoramento tenham evoluído juntamente com as redes definidas por software, melhorando consideravelmente a flexibilidade das arquiteturas de medição, os métodos de coleta de fluxo e amostragem utilizados, ainda se mantém os mesmos empregados nas técnicas tradicionais de medição de redes (TAN et al., 2021). Neste contexto, a *In-band Network Telemetry* (INT), vem sendo apresentada como uma alternativa recente ao encaminhamento de informações a coletores externos.

2.2.3 Monitoramento INT

A *In-band Network Telemetry* é um paradigma emergente de monitoramento em infraestruturas de rede que busca obter uma maior visibilidade do ambiente, proporcionando melhor escalabilidade, precisão, cobertura e desempenho. Ao contrário do moni-

toramento tradicional e do realizado nas redes SDN, a INT obtêm o *status* do ambiente realizando a inserção de metadados pelos nós da infraestrutura, combinando a medição de rede com o encaminhamento de pacotes, possibilitando ainda a tomada de decisões diretamente no plano de dados (TAN et al., 2021).

Basicamente, por meio da programação implementada no plano de dados e utilizando-se dos fluxos ativo da rede, torna-se possível realizar o armazenamento de informações sobre o estado do ambiente. A medida que os pacotes trafegam por dispositivos com suporte a INT, as informações são coletadas, sendo extraídas e tratadas ao final do trajeto (CAMERA, 2020). Alternativamente, análises podem ser realizadas diretamente sobre os fluxos de dados que transitam pelos nós INT, sem a necessidade de inclusão de informações e, independente da forma adotada, ações prévias podem ser tomadas diretamente no plano de dados sem que sejam necessários encaminhar para um sistema externo. Na Seção 2.2.4 a telemetria de rede *in-band* é abordada de forma mais aprofundada.

2.2.4 *In-band Network Telemetry*

Segundo Yu (2019), a telemetria em rede é um processo automatizado para coletar e processar informações da rede, tendo como objetivo, segundo Tan et al. (2021), entender o que está acontecendo no ambiente. As informações coletadas são utilizadas como base para uma grande variedade de tomada de decisões que objetivam melhorar o desempenho, disponibilidade, segurança e eficiência da rede.

Este processo pode também ser definido como um *framework* projetado para permitir a coleta e emissão de relatórios sobre o estado da rede, sendo realizado por intermédio do plano de dados, sem necessidade de intervenção ou realização de ações pelo plano de controle. Na arquitetura INT, *headers* (cabeçalhos de pacotes) adicionais podem ser adicionados aos pacotes, permitindo que dispositivos de rede programáveis (INT *transit*) interpretem essas informações. Assim, o plano de dados pode ser programando para que fluxos específicos de rede sejam detectados e tratados (GROUP, 2020). Alguns exemplos de aplicação deste *framework* são:

- OAM (*Operation Administration and Maintenance*): as informações do estado da rede são coletadas no pano de dados e repassadas para um controlador externo.

- Controle em tempo real: as informações coletadas no plano de dados podem ser repassadas para a origem do tráfego, que de posse destes dados tem a possibilidade de alterar o encaminhamento dos pacotes se necessário.

- Detecção de evento de rede: os coletores podem gerar ações para responder a eventos de rede, tão logo alguma condição especial seja detectada, tal como um congestionamento, e necessite de tratamento imediato.

Ademais, embora a telemetria *in-band* tenha sido abordada pela primeira vez no P4.org recentemente, em 2015, uma considerável quantidade de variações vem sendo estudadas e discutidas no IETF (*Internet Engineering Task Force*) e pela indústria. Atualmente, existem três diferentes modos de operação definidos na especificação INT (GROUP, 2020):

- **INT-XD** (*eXport Data*): os metadados são diretamente exportados com base nas instruções do plano de dados, sem modificação do pacote.
- **INT-MX** (*eMbed instruct(X)ions*): instruções são inseridas no nó inicial e os demais nós encaminham metadados ao sistema de monitoramento, tendo como base as instruções que foram incluídas no cabeçalho. Por fim, o último nó remove as instruções antes de encaminhar o pacote ao destinatário.
- **INT-MD** (*eMbed Data*): neste modo, além das instruções, os metadados também são adicionados ao pacote e no último nó, instruções e metadados são removidos, sendo seletivamente enviados ao sistema de monitoramento.

A Figura 7(a) ilustra o funcionamento do modo INT-XD, mediante um fluxo f_n que percorre um caminho entre os dispositivos A e F. Como as instruções estão configuradas diretamente no plano de dados, os espaços disponíveis nos pacotes não são utilizados. Ao contrário, cada pacote tem suas informações analisadas com base nestas instruções e conforme apresentado nos passos (1),(4) e (6), os dados de telemetria são diretamente exportadas para o coletor.

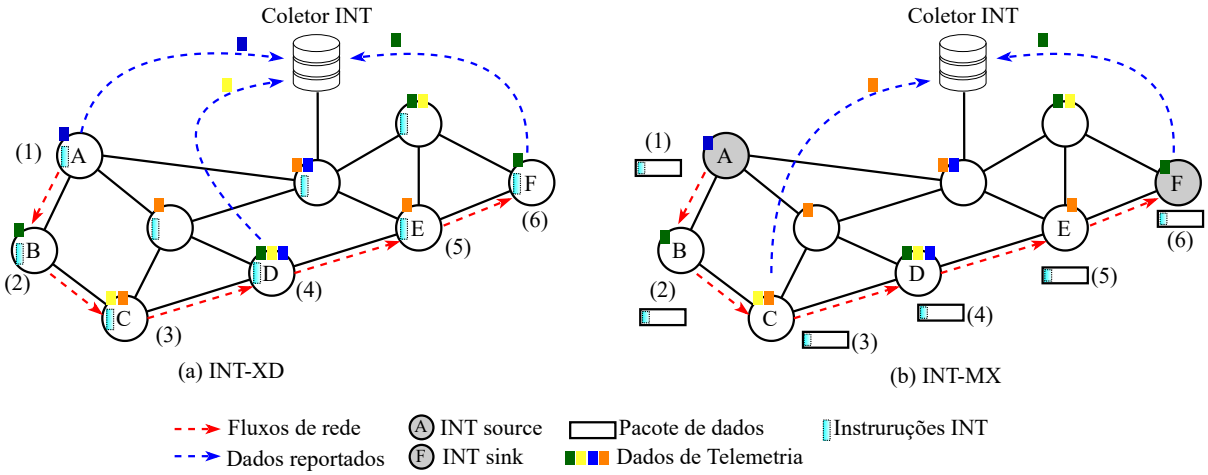


Figura 7 – Modos de operação INT-XD e INT-MD

A abordagem INT-MX é ilustrada na Figura 7(b), utilizando o mesmo fluxo e caminho anterior. Neste caso, somente as instruções são encapsuladas no pacote e em cada dispositivo elas são desencapsuladas, executadas e encapsuladas novamente. Os dados de telemetria resultantes destas execuções são então reportados para o coletor,

conforme apresentado nos passos (3) e (6). Por fim, o modo de funcionamento INT-MD foi apresentado na seção 1.1, ilustrado na Figura 1.

2.3 Dispositivos Programáveis

O surgimento da arquitetura SDN trouxe uma demanda por novos dispositivos capazes de serem reconfigurados, possibilitando por exemplo a implementação de novas funções e protocolos. Interfaces como SmartNIC (*Smart Network Interface Card*) e NetFPGA (*Network Field-Programmable Gate Array*) são alguns dos dispositivos capazes de atender a essa demanda, sendo abordados mais detalhadamente nesta seção.

2.3.1 SmartNICs Netronome

Uma NIC (*Network Interface Card*) é uma interface de rede tradicional, ou seja, um dispositivo que fornece funcionalidades pré-definidas sem a possibilidade de alteração ou personalização. Já as SmartNICs ou NICs inteligentes, podem se consideradas interfaces completamente programáveis. A arquitetura destas interfaces disponibilizadas pela Netronome conta com um processamento paralelo e multi-core baseado em uma arquitetura *systen-on-chip* (SoC), a qual combina uma ou mais CPUs com as funções padrões da NIC (MIANO et al., 2019), suportando velocidades de até 100Gbps.

A arquitetura da SmartNIC NFP4000 da Netronome (NETRONOME, 2016) – a qual é utilizada neste trabalho – é apresentada na Figura 8. Esta interface apresenta uma arquitetura modular, sendo composta por inúmeras ilhas, tais como ilhas de memórias, unidades de processamento e controle de endereço MAC (*Media Acess Controll*). As unidades de processamento são máquinas de 32 *bits* denominadas FPC (*Flow Processing Cores*), as quais possuem espaços distintos para armazenamento de dados e instruções, permitindo que cada unidade execute um conjunto diferenciado de instruções, contando ainda com 8 *threads* que executam sobre o mesmo código e memória local.

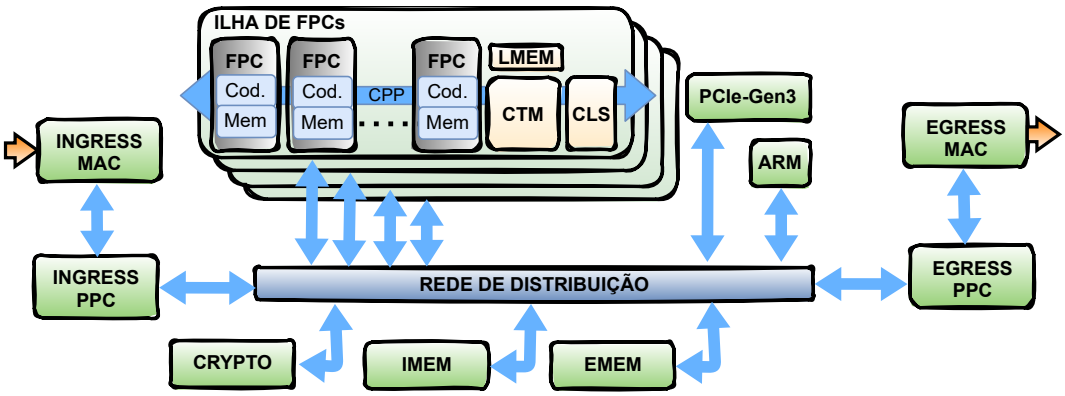


Figura 8 – Arquitetura NFP4000

Como o processador de uma FPC é uma máquina de 32 *bits*, todos os seus registradores internos e memória local são formados por palavras de 32 *bits* e a troca de

informações entre as FPCs, registradores e memórias são realizadas por um barramento CPP (*command-push-pull*). Cada FPC é composta ainda por:

- 256 registradores de propósito geral, divididos entre as suas 8 *threads*, resultando em 32 registradores para cada uma.
- 512 registradores de transferência, utilizados para trafegar os dados no barramento CPP entre FPCs de ilhas vizinhas ou memórias externas a ilha da FPC.
- 128 registradores *next-neighbor*, utilizados para comunicar com FPCs vizinhas na mesma ilha.
- memória local com 1024 palavras de 32 bits compartilhadas entre as*threads*

A interface NFP4000 possui 5 ilhas de processamento compostas por 12 FPCs e uma memória local (LMEM) de 4 KB, podendo possuir ainda outros dois tipos de memória: *Cluster Local Scratch* (CLS) de 64 KB, utilizada para dados necessários para maioria dos pacotes ou compartilhamento de pequenas tabelas e *Cluster Target Memory* (CTM) de 256 KB, utilizada cabeçalhos de pacotes e coordenação entre FPCs e outros subsistemas. Externamente as ilhas de processamento, cada interface possui ainda mais duas memórias SRAM: uma *Internal Memory* (IMEM) de 4 MB, destinada ao corpo dos pacotes e compartilhamento de tabelas médias e duas *External Memory* (EMEM) de 3 MB, para compartilhamento de grandes tabelas (NETRONOME, 2016; WRAY, 2014).

Cada pacote recebido em uma FPC é diretamente alocado a uma *thread*, assim cada interface tem a capacidade de processar 480 pacotes simultâneos. A ilha MAC é responsável por realizar a validação dos pacotes e armazenar em um *buffer*[9] intermediário. A ilha PPC (*Packet Processing Cores*), que possui uma interface direta com a ilha MAC, é responsável pela distribuição dos pacotes para cada FPC. A Figura 9 demonstra o processo de distribuição dos pacotes para cada ilha de FPCs. Nesta ilha, cada pacote é direcionado para uma FPC, mediante da realização de um *hash* com base nas informações de seu cabeçalho.

A programação destas interfaces pode ser realizada com linguagens de alto nível como P4 e Micro-C (NETRONOME, 2017) através de um compilador proprietário, apresentando, no entanto, algumas restrições e diferenças comportamentais. No P4 disponibilizado, os campos de metadados são mais restritos quando comparados com a especificação da linguagem e alguns comportamentos são distintos dos obtidos em emuladores, devido a arquitetura paralela adotada. Ainda, a linguagem Micro-C disponibilizada consiste em um subconjunto da linguagem C, principalmente devido a limitações relacionadas aos tipos de dados suportados, funções e cabeçalhos de biblioteca padrão.

[9] Uma região de memória física utilizada para armazenar temporariamente os dados enquanto eles estão sendo movidos de um lugar para outro.

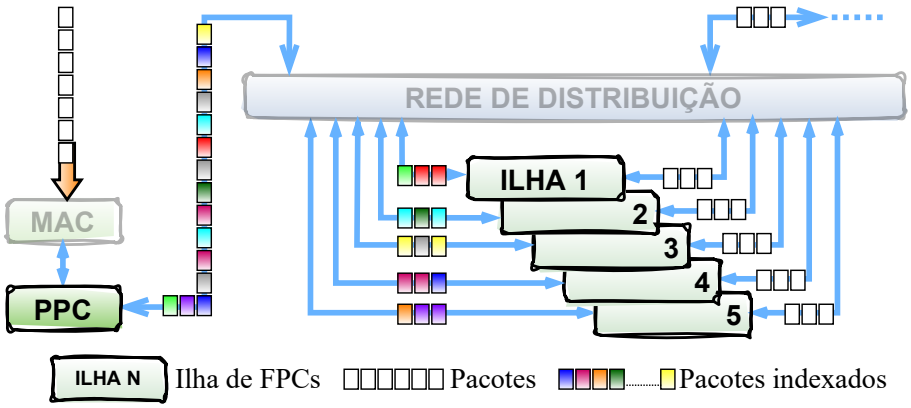


Figura 9 – Distribuição de pacotes para ilhas de FPCs

2.3.2 NetFPGA

A NetFPGA é uma plataforma aberta de *hardware* e *software* que combina um FPGA (*Field-Programmable Gate Array*[10]), memórias SRAM e DRAM e processadores de sinais. Sendo voltada para área de pesquisa e ensino, tem por objetivo viabilizar o desenvolvimento de protótipos de sistemas de rede de alta velocidade, permitindo a modificação dos pacotes em trânsito, controle sobre o encaminhamento e manutenção de estado (NETFPGA, 2013). Esta plataforma se destaca como uma alternativa de *hardware* flexível e de baixo custo devido a possibilidade de atingir altas taxas de encaminhamento, mantendo uma baixa latência, sem comprometer a largura de banda e o processamento do *host*[11] (GOULART et al., 2015).

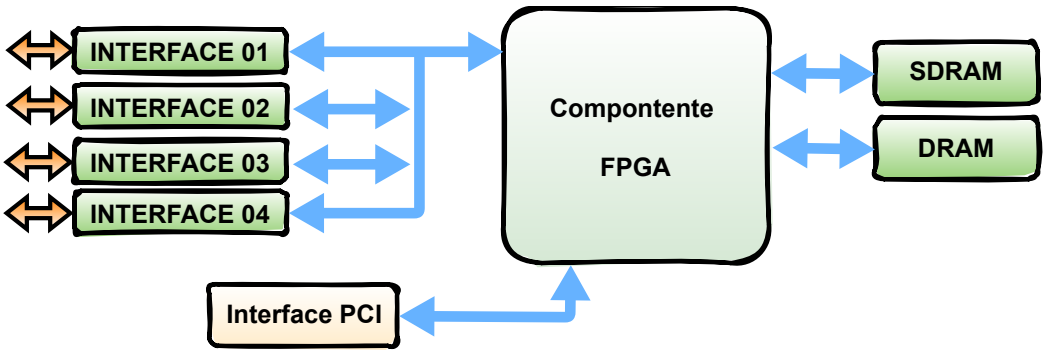


Figura 10 – Arquitetura NetFPGA

Tais características são obtidas devido ao encaminhamento de pacotes realizado em *hardware*, onde os fluxos são processados imediatamente ao serem recebidos pela interface. Como resultado, estas interfaces apresentam aumento significativo de desempenho em relação a dispositivos de encaminhamento com OpenFlow emulado, permitindo também uma aproximação de um ambiente mais equivalente a um ambiente real (ELLER, 2016).

[10] Um arranjo de portas programável em campo, um circuito integrado projetado para ter suas funcionalidades definidas pelos usuários (um *hardware* programável).
[11] Qualquer dispositivo conectado a rede

A arquitetura básica da plataforma NetFPGA, apresentada na Figura 10 consiste em um FPGA central responsável pela lógica de processamento dos pacotes e pela gerencia dos demais componentes, um conjunto de quatro interfaces Ethernet, uma interface PCI para comunicação com o *host* e o conjunto de memórias SRAM e DRAM.

Tabela 1 – Configurações das placas NetFPGA disponíveis

	10G	**1G-CML**	**SUME**
Ethernet:	4x 10Gbps (SFP+)	4x 1Gbps	4x 10Gbps (SFP+)
SRAM (MB)	27	4.5	27
DRAM (MB)	RLDRAM II 288	DDR3 512	DDR3 8192
PCI	Express Ger 1	Express Ger 2	Express Ger 3
FPGA	Xilinx Virtex-5	Xilinx Kintex-7	Xilinx Virtex-7

Atualmente os modelos disponíveis são NetFPGA-1G-CML com interfaces Ethernet de 1 Gbps, NetFPGA-10G com interfaces SFP+ (*Samll Form Factor Pluggable Pluss*[12]) que suportam até 10 Gbps e a mais recente, NteFPGA-SUME, também com interfaces SFP+ para até 10 Gbps. Esta interface mais recente apresenta também suporte a uma largura de banda de até 100 Gbps, possibilitada pelo sistema de interfaces seriais adicionais. Por meio desta expansão, podem ser adicionadas módulos FMC (FPGA *Mezzanine Card*) adicionais realizando a expansão da capacidade inicial da largura de banda (ZILBERMAN et al., 2014). A Tabela 1, apresenta as configurações de cada uma destas interfaces.

Para a programação das NetFPGAs, geralmente se faz uso de linguagens do tipo HDL (*Hardware Description Languages*), sendo as linguagens Verilog e VHDL as mais difundidas. No entanto, com a adoção de um conjunto de ferramentas denominado Xilinx P4-SDNet é possível obter um ambiente de desenvolvimento para programação diretamente com a linguagem P4, sem a necessidade de conhecimento em HDL (NETFPGA, 2018).

[12] Modelo de interface de rede que suporta velocidades de até 10 Gbps

3 TRABALHOS RELACIONADOS

A utilização da programabilidade no plano de dados juntamente com o paradigma INT tem atraído o interesse da academia e da indústria. Exemplos de trabalhos na área incluem a análise de sobrecarga, mitigação de ataques, a classificação de fluxos de dados, e otimização do caminho para coleta de informações. Neste capítulo, apresenta-se os principais trabalhos que abordam o monitoramento de rede no plano de dados baseados em INT. Inicialmente são apresentados os trabalhos que abordam o plano de dados com uma menor enfase, seguido pelos trabalhos e focam na análise diretamente no plano de dados. Por fim são abordados os trabalhos que mais se assemelham ideia proposta.

Os trabalhos seguintes se aproximam levemente as características da abordagem proposta, a medida que são desenvolvidos sobre um ambiente físico, embora ainda apresentem uma menor enfase no PD. O mecanismo de previsão de atraso de fila em tempo real apresentado por Feng et al. (2019) faz uso de um ambiente físico, composto por dispositivos de encaminhamento programáveis do modelo Wedge 100BF. A análise de todos os pacotes que transitam pela rede é realizada pelo plano de dados, sendo tratada como pré-processamento, consistindo em registrar e agrupar as informações. Os dados são então repassados a um coletor externo, que realiza o tratamento e apresentação das informações.

A utilização de SmartNICs para desenvolvimento de uma plataforma de telemetria de rede é apresentada em Feng et al. (2020). Nesta implementação todos os pacotes são analisados em cada nó da rede, adotando a análise de um conjunto de campos. Para tal são utilizados limites estáticos pré-definidos para cada campo, cuja forma como são definidos não é abordada claramente. É implementado também um dispositivo coletor em P4 para análise dos cabeçalhos INT. Um algoritmo em Micro-C é executado no *host* de monitoramento, para tarefas mais complexas, tais como agregação, detecção de evento, notificação, etc.

A implementação de uma metodologia INT-MD sobre SmartNICs da Netronome é realizada no Int Flow (CAMERA, 2020), utilizando um pacote adicional, denominado sonda. Este pacote é injetado na rede especificamente para realizar a coleta de metadados, sendo posteriormente encaminhado para um coletor, responsável pelo tratamento e exibição das informações. Para identificar os pacotes do tipo sonda, é realizada uma marcação no campo *diffServ* do cabeçalho IPv4, recebendo assim a inserção de cabeçalhos INT adicionais no início da topologia e atualizadas no decorrer do trajeto, sendo removidas no último *switch*, antes de ser entregue para o destino.

O desenvolvimento de uma ferramenta denominada NetSeer, é apresentado por Zhou et al. (2020), utilizando um ambiente com dispositivos de encaminhamento da Barefoot e SmartNICs da Netronome. Nela é apresentada a detecção de eventos de fluxo sobre todos os pacotes da rede. As informações são coletadas a partir dos valores disponíveis nos pacotes, sem a inclusão de informações adicionais nos fluxos. Os dados coletados são então compactados e enviados para uma análise e processamento externamente ao

plano de dados.

A coleta e o encaminhamento das informações para uma análise externa ainda podem ser observados em outros trabalhos que tem como foco o monitoramento de rede com abordagens distintas. A avaliação do comportamento do tráfego da rede com uso da coleta de informações (LIN et al., 2021) e a implementação de monitoramento seletivo e dinâmico com o uso de cabeçalhos adicionais (SUH et al., 2020), são alguns destes trabalhos. No entanto, todas as propostas já mencionadas impõem uma menor enfase no uso do plano de dados, visto que o processamento das informações é realizado de forma externa. Tais trabalhos ainda se diferenciam desta proposta por não tratarem outros pontos como a aplicação de métricas dinâmicas visando acompanhar as mudanças dos fluxos e a utilização de uma média móvel para amenizar as variações ocorridas.

No entanto, alguns trabalhos, tal como os que serão apresentado a seguir, passam a imputar uma maior enfase ao plano de dados, realizando análises mais aprofundadas nas informações dos datagramas. Em Kim, Suh e Pack (2018) uma abordagem de telemetria seletiva é implementada sobre um ambiente emulado. A implementação realizada diretamente no plano de dados, tem como objetivo reduzir a sobrecarga no monitoramento da rede em comparação com o uso de uma abordagem INT Clássica. Cabeçalhos adicionais são inseridos em pacotes selecionados com base em uma taxa de inserção. Esta taxa é armazenada estaticamente em tabelas indexadas pelos identificadores de fluxo e a definição de quais pacotes serão analisados é realizada com base em valores pré-definidos.

Por sua vez, o LINT (CHOWDHURY; BOUTABA; FRANÇOIS, 2021) é um mecanismo que busca adaptar a precisão da telemetria com a sobrecarga gerada na rede. Em cada nó da rede, analisa o impacto de não realizar o envio das informações para o coletor, decidindo com base em uma função preditora e métricas estáticas, se adiciona ou não os dados. A função preditora utiliza média móvel exponencial sobre os itens do fluxo, analisando cada campo INT individualmente e realizando uma comparação com base nos valores do dispositivo de encaminhamento anterior e do atual.

O trabalho apresentado por Mai et al. (2021), utiliza um ambiente emulado para desenvolvimento de uma arquitetura de controle de rede, onde os dispositivos se adaptam automaticamente. No plano de dados é realizada a coleta de informações com a adoção de um pacote sonda inserido na rede. Os dados coletados são então encaminhados para tomada de decisão pelo plano de controle. Dessa forma, a definição das ações a serem realizadas no plano de dados é realizada pelo plano de controle, com base nas informações recebidas.

Na aplicação denominada FlowStalker (CASTANHEIRA; PARIZOTTO; FILHO, 2019), é apresentado um mecanismo de classificação de fluxo em duas etapas, no modelo de monitoramento INT-XD. Em uma primeira etapa, denominada proativa, é realizada uma contagem de todos os pacotes. Já na segunda etapa, denominada reativa, é realizado um monitoramento sobre os fluxos ou pacotes que foram selecionados na etapa anterior.

Ainda, implementa também uma clusterização referente a rotas, a fim de descentralizar a coleta de informações.

Na estratégia P4 Entropy (DING; SAVI; SIRACUSA, 2020), são implementado a realização de cálculos logaritmos e exponenciais diretamente no plano de dados. Tais cálculos são utilizados para obter a entropia do tráfego, fazendo uso das informações contidas nos pacotes, utilizando limites pré-definidos para apuração da entropia. No trabalho, a maior contribuição são os cálculos implementados, sendo os resultados da entropia encaminhados ao controlador para registro, não sendo aplicadas tomadas de decisões ou ações complementares.

A solução denominada SwitchTree (LEE; SINGH, 2020) apresenta a adoção do algoritmo Random Forest no plano de dados. A implementação faz uso de valores pré-definidos em tabelas de *match-action*, que são utilizadas para análise dos pacotes e tomada de decisão. Os valores utilizados como parâmetros de comparação são definidos a partir de uma etapa de treinamento, sendo possível atualizar em tempo de execução por meio do plano de controle. Assim, mesmo que com um maior foco no plano de controle, os trabalhos apresentados ainda se utilizam de ambientes emulados, e em sua maioria, com limiares de comparação estático. Ademais, outros pontos como a inclusão de dados na rede, a análise parcial dos pacotes trafegados e a ausência de análise de dados históricos, diferenciam alguns destes trabalhos da propostas apresentada.

No entanto, alguns poucos trabalhos que enfatizam o plano de dados, foram executados sobre ambiente reais. A implementações de coletores de estatísticas com objetivo de corrigir a prioridade dos pacotes é desenvolvida com dispositivos de encaminhamento da Barefoot (CUGINI et al., 2019). Um mecanismo de detecção de eventos INT programável, implementado em interfaces SmartNIC da Netronome é apresentado em Vestin et al. (2019). Por fim, e um *framework* escalável com três tipos de monitoramento denominado IFIT (LU et al., 2019) é implementado em um roteador CX600-M2H da Huawei, com da utilização de microcódigo.

Contudo, mesmo fazendo uso de um ambiente físico, estes trabalhos ainda encontram limitações como a análise parcial dos pacotes e a inclusão de informações adicionais na rede. Ademais outros fatores como utilização de limiares estáticos e ausência de análise de dados históricos estão presentes em parte destes trabalhos, fazendo com que não se igualem a proposta testa dissertação Dessa forma, embora uma considerável gama de trabalhos relacionados tenha sido analisada e apresentada, é possível perceber que todos apresentam distinções com relação a implementação proposta e vários pontos. Assim sendo, restringem-se a quatro as abordagens que apresentem uma maior proximidade com o trabalho proposto, sendo demonstradas a seguir.

A proposta de classificação de pacotes em classes, implementada com P4 sobre interfaces FPGAs é apresentada em Xiong e Zilberman (2019). A classificação ocorre por meio de algoritmos de aprendizado de máquina mapeados para tabelas de *match-action*,

aplicadas sobre as informações presentes nos cabeçalhos. No entanto, embora analise todos os pacotes, os limiares de comparação utilizados são definidos estaticamente no plano de dados. Ainda, outros pontos como ausência da análise de dados históricos e processamento das informações diretamente no plano de dados, não são implementadas neste trabalho.

A implementação proposta por Lapolli, Marques e Gaspary (2019) apresenta um mecanismo de detecção de ataques DDoS em tempo real. O desenvolvimento é realizado sobre um ambiente emulado, utilizando a linguagem P4 com o objetivo de ser executado inteiramente no plano de dados. Além das limitações do ambiente, a desconsideração dos valores históricos é o único ponto distinto, mas que talvez não seja relevante para a detecção de ataques de negação de serviço.

No enanto, apesar de possuir características semelhantes, o trabalho de Lapolli, Marques e Gaspary (2019) considera somente a entropia dos fluxos. O uso de informações mais aprofundadas, tais como o tempo de permanência no dispositivo e o tamanho do pacote, abordadas nesta dissertação. Ademais, tem como foco o monitoramento e detecção de um tipo específico de problema, sendo mais restrito que a proposta apresentada, que visa identificar anomalias sem restrição de tipo.

Em Xavier et al. (2021), é apresentado um modelo de aprendizado de máquina mediante um algoritmo de árvore de decisão diretamente no plano de dados. A implementação é realizada com o mapeamento dos algoritmos para estruturas de controle, similar ao apresentado em outros trabalhos (XIONG; ZILBERMAN, 2019; LEE; SINGH, 2020). O objetivo da abordagem é demonstrar a viabilidade de implantação destes algoritmos, realizando a classificação por pacotes ou fluxos, para um servidor de detecção de intrusão. Com base em um conjunto de dados de aprendizado, uma aplicação realiza a implementação da estrutura de árvore de decisão fazendo uso de estruturas condicionais *if-else* aninhadas. Posteriormente, a estrutura criada é mapeada para a linguagem P4, sendo assim realizada a classificação de fluxos ou pacotes.

Ainda, o trabalho realiza uma análise em todos os pacotes quando adota esta opção de monitoramento, não realizando a inclusão de informações adicionais, tal como o trabalho proposto. No entanto, segundo a própria avaliação dos autores, mesmo que utilize dados reais para obter os limiares de comparação, estes não se atualizam dinamicamente, desconsiderando a dinamicidade de um tráfego de rede. Outro ponto a ser observado consiste na utilização de valores históricos que não é levada em consideração devido as limitações encontradas na linguagem. Com relação ao desempenho, os resultados deste trabalho demonstram que quando realizada na análise de fluxo, existe o risco de consumo de memória excessivo. Dessa forma, embora apresente características muito próximas, a implementação de Xavier et al. (2021) não tem o mesmo viés que o trabalho proposto, tanto pelas características como pelo objetivo.

Por fim, no trabalho apresentado por Dimolianis, Pavlidis e Maglaris (2020),

desenvolve-se uma arquitetura para detecção de ataques DDoS implementada sobre SmartNICs da Netronome. É realizada a combinação de importantes métricas de tráfego malicioso, envolvendo o número de fluxos e à simetria do pacote, com objetivo de identificar anomalias. Os alarmes apropriados são disparados em períodos baseados no tempo e transmitidos para sistemas de mitigação externos. Essa abordagem é realizada diretamente no plano de dados, sob informações disponíveis em todos os fluxos de dados, relacionando a quantidade de fluxos com a rede monitorada.

Este trabalho pode ser considerado o mais próximo a proposta apresentada, devido a grande similaridade que apresenta com relação as características analisadas. No entanto, o mesmo é aplicado com foco na detecção de ataques em redes específicas, as quais são denominadas redes monitoradas, levando em consideração a quantidade de pacotes para classificar um ataque. Diferentemente, a proposta apresentada não tem como foco redes específicas e busca aplicar a sua análise sobre todo o tráfego existente na rede. Ademais, embora este trabalho também faça uso de uma média móvel exponencial, a mesma é aplicada para classificação os fluxos em épocas de tempo buscando uma avaliação quantitativa, além de ser necessário o redirecionamento dos fluxos para o seu monitoramento.

Como é possível verificar por intermédio dos trabalhos analisados, várias propostas se utilizam do plano de dados para realizar algum nível de monitoramento baseado no paradigma INT. Em grande parte das propostas, não é realizam nenhum processamento no plano de dados, servido este apenas para encaminhar as informações para análises externas. Ainda, algumas poucas se assemelham mais a ideia proposta, utilizando o plano de dados para realizar algum processamento.

No entanto, além dessa similaridade, a grande maioria destas aplicações não possuem muitas outras semelhanças com as técnicas empregadas neste trabalho. Ademais, muitas delas, principalmente as mais semelhantes, são voltadas para domínios específicos de aplicação. Dessa forma, é possível afirmar que este trabalho apresenta significativas contribuições que não foram ainda analisadas. A Tabela 2, apresentada a seguir exibe uma comparação resumida do que foi apresentado neste capítulo, considerando:

1. Utilização de *hardware* físico
2. Utilização de limiares de comparação dinâmicos
3. Não inclusão de dados na rede
4. Análise de todos os pacotes
5. Utilização dos valores históricos (fazendo uso de uma Média Móvel Exponencial)
6. Análise de informações diretamente no plano de dados
7. Uso de tráfego elevado, superior a 10Gbps

Tabela 2 – Trabalhos relacionados e itens analisados

Trabalho \ Item	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Feng et al. (2019)	X			X			
Feng et al. (2020)	X			X			
Camera (2020)	X						
Zhou et al. (2020)	X		X	X			
Lin et al. (2021)				X			
Suh et al. (2020)			X				
Kim, Suh e Pack (2018)						X	
Chowdhury, Boutaba e François (2021)				X	X	X	
Mai et al. (2021)		X				X	
Castanheira, Parizotto e Filho (2019)			X			X	
Ding, Savi e Siracusa (2020)			X	X		X	
Lee e Singh (2020)			X	X		X	
Cugini et al. (2019)	X					X	
Vestin et al. (2019)	X				X	X	
Lu et al. (2019)	X	X				X	
Xiong e Zilberman (2019)	X		X	X			X
Lapolli, Marques e Gaspary (2019)			X	X	X	X	
Xavier et al. (2021)	X		X	X		X	
Dimolianis, Pavlidis e Maglaris (2020)	X	X	X	X	X	X	
APF - abordagem proposta	**X**	**X**	**X**	**X**	**X**	**X**	**X**

4 NOTIFICAÇÃO SELETIVA DE DADOS INT EM SMARTNICS

Neste capítulo, detalha-se a proposta de trabalho desta dissertação, sendo apresentada inicialmente através de uma abordagem mais ampla. Dessa forma, objetiva-se viabilizar além da compreensão do funcionamento de um modo geral, um entendimento sobre a definição dos valores utilizados. Em seguida é apresentada a estrutura tradicional de encaminhamento de pacotes para um plano de dados em P4. Posteriormente é então abordada a implementação proposta para o uso em interfaces programáveis. Por fim, discute-se as limitações encontradas e suas implicações.

4.1 APF - Analisador preliminar de fluxo

O trabalho proposto adota o INT-XD como modo de operação, cujo escopo é apresentado na Figura 11. O objetivo principal consiste em realizar uma análise em todos os pacotes p que transitam por um dispositivo de encaminhamento programável d, pertencente a um ambiente D ($d \in D$). Para cada pacote, é realizado em d, o processamento de um conjunto de métricas disponíveis N, mediante a aplicação de uma Média Ponderada (MP). A MP obtida é então comparada com uma média histórica obtida por meio de uma Média Móvel Exponencial Ponderada (MMEP) a fim de classificar o comportamento do fluxo de dados. Dessa forma, é realizada a definição dos pacotes que devem ser encaminhados para o coletor INT.

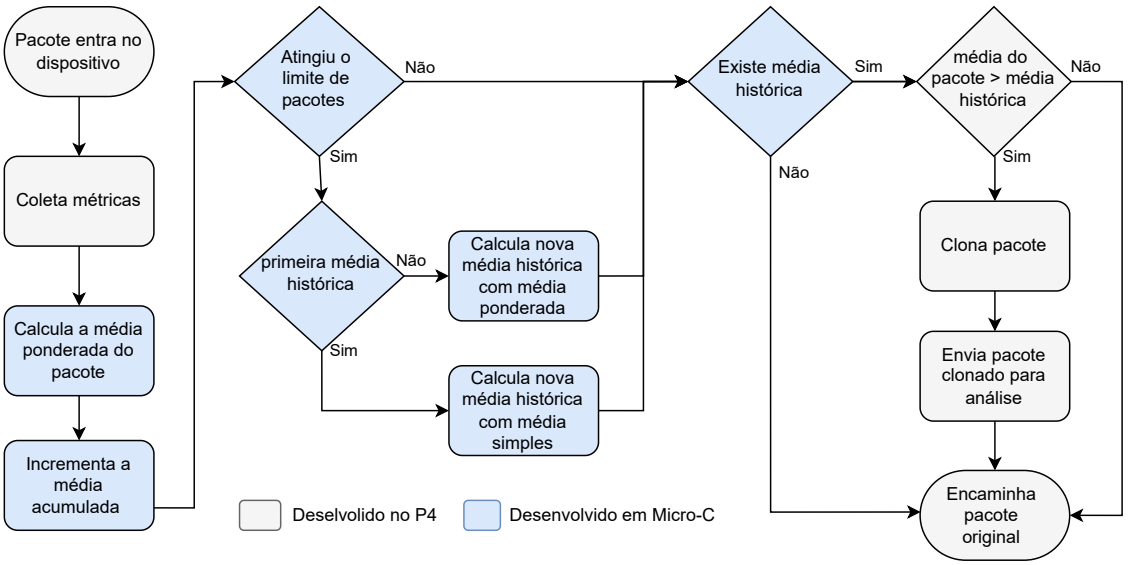


Figura 11 – Fluxograma do escopo do trabalho proposto.

Ao contrário da média simples, onde o somatório dos valores é dividido pela quantidade de termos, em uma média ponderada, um peso (P_n) é atribuído para cada termo (T_n), sendo o resultado obtido pelo somatório das multiplicações entre termos e pesos, divido pelo somatório dos pesos. A Equação (4.1), exemplifica a obtenção de uma média ponderada com n termos.

$$MP = \frac{\sum_{i=1}^{n} T_i \cdot P_i}{\sum_{i=1}^{n} P_i}. \tag{4.1}$$

No ambiente adotado, os termos utilizados para obtenção da média ponderada foram provenientes do conjunto de metadados $\mathbb{N}^+$, disponibilizados pela linguagem P4, o qual possui uma relevante gama de informações. Alguns exemplos dessas informações são o *timestamp* de ingresso e saída dos blocos de controle, tempo de espera na fila, tamanho do pacote e tamanho da fila de entrada e saída, entre outros. No entanto, conforme as próprias especificações da linguagem, cada desenvolvedor pode realizar modificações no seu modelo de referência e compilador, adicionando ou removendo campos. Esta personalização ocorre efetivamente com a arquitetura disponibilizada pela Netronome, a qual apresenta algumas diferenças com relação ao P4 original, limitando a quantidade de metadados disponíveis.

Assim, o conjunto de métricas $N \in \mathbb{N}^+$ utilizado, é composto somente pelo tamanho do pacote (t_p) e a latência (l_p) obtida em nanossegundos (ns), obtida pela diferença entre o *timestamp* atual e o de ingresso no dispositivo. De posse destas duas métricas, é realizado o cálculo da MP do pacote (m_p), atribuindo-se um maior peso para latência e um peso inferior para o tamanho do pacote, conforme a expressão demonstrada na Equação (4.2). No entanto, diferentemente da Equação (4.1), a divisão pela soma dos pesos não é necessária, visto que seu resultado é igual a um.

Com isso, no decorrer de um intervalo pré-definido de pacotes J, denominado janela, é realizado o somatório das m_p, resultando em uma média ponderada acumulada para janela (mpa). Ao final de cada intervalo, que equivale a quantidade de pacotes cujas médias foram somadas em mpa, primeiramente é realizado o cálculo da média da janela (m_J), conforme a Equação (4.3). Posteriormente, para cada intervalo J, é recalculado o valor da média histórica (m_h), realizado de duas formas distintas, considerando a existência ou não de uma média anterior.

$$m_p = l_p * 0.7 + t_p * 0.3 \tag{4.2}$$

$$m_J = \frac{\sum_{i=1}^{J} m_p}{J} \tag{4.3}$$

$$m_h = m_J \tag{4.4}$$

No primeiro caso, a média histórica é equivalente a média da janela, conforme Equação (4.4), sendo aplicada somente na ocorrência do primeiro intervalo J_1, onde não existe uma média histórica definida, para que seja realizada uma ponderação entre ambas. O segundo caso é utilizado em todas as demais atualizações, aplicando uma MMEP, um indicador de tendência que leva em consideração o valor do período anterior (v_{i-1}) e do atual (v_i). Ainda, ao valor de cada período, são aplicados pesos distintos, definindo assim

a importância de cada um no resultado final. Este peso é denominado multiplicador de ponderação (λ), onde $0 \leq \lambda \leq 1$. Dessa forma, quando mais próximo a um for o λ, maior será a importância dada aos valores do período a ele relacionado. Na Equação (4.5), o λ é aplicado sobre v_{i-1}, dessa forma, quanto mais próximo a um for o valor de λ maior será a importância do período anterior e consequentemente, menor a importância do período atual (v_i).

$$MMEP = v_{i-1} * \lambda + v_i * (1 - \lambda) \tag{4.5}$$

$$m_h = m_h * \lambda + m_J * (1 - \lambda) \tag{4.6}$$

Assim, após a obtenção da primeira média, as demais médias históricas são calculadas com a adoção da expressão demonstrada na Equação (4.6). Dessa forma, a partir de uma média histórica inicial definida, os pacotes passam a ser analisados encaminhados ao coletor, sempre que a média do pacote for superior a média histórica ($m_p > m_h$). Tais pacotes são então duplicados, possibilitando que sejam analisados ao mesmo tempo em que os pacotes originais são encaminhados ao seu destino, sendo a cópia do pacote enviada para o coletor.

4.2 Encaminhamento básico de pacotes

A PPD se inicia pela definição dos cabeçalhos a serem analisados, que definem o tipo de pacote que será passível de analise neste plano. Por se tratar de um ambiente experimental cujo foco se encontra na análise e avaliação do tratamento de fluxos diretamente no plano de dados, os cabeçalhos criados ficaram restritos aos mais relevantes a uma comunicação padrão, sendo criados os cabeçalhos Ethernet, IPv4 e UDP. A Figura 12 apresenta a codificação destes cabeçalhos realizada no programa P4, juntamente com a definição da estrutura que os engloba, denominada *struct headers*.

Após a definição dos cabeçalhos, o *parser* é programado para realizar a extração dos dados do pacote, cuja sequência implementada é demonstrada na Figura 13. O fluxo principal de um pacote se inicia extraindo o cabeçalho *ethernet* e verificando o tipo de pacote presente, sendo encaminhando para inicialização do cabeçalho IPv4, caso corresponda. Após extrair as informações para estrutura *header ipv4_t*, o *parser* verifica a existência de um cabeçalho UDP, encaminhando para extração dos dados para estrutura *header udp_t* em caso positivo. Caso o pacote não seja do tipo Ethernet, o mesmo não é analisado pelo plano de dados. Para os demais estados do *parser*, pode ocorrer a não correspondência com a verificação que está sendo realizada, o que faz com que o pacote seja então aceito e encaminhando para a próxima etapa.

Após a inicialização dos cabeçalhos, é realizada a verificação do *checksum* do pacote por meio da implementação padrão da linguagem, validando a corretude do mesmo, sendo então encaminhando para o *pipeline* de *match+action*. No *ingress*, primeiro bloco de

```
header ethernet_t {
    var48_t    dstAddr;
    var48_t    srcAddr;
    bit<16>    etherType;
}

header udp_t {
    bit<16>    srcPort;
    bit<16>    dstPort;
    bit<16>    lengthUdp;
    bit<16>    checksum;
}
```

```
header ipv4_t {
    bit<4>     version;
    bit<4>     ihl;
    bit<8>     diffServ;
    bit<16>    totalLen;
    bit<16>    identification;
    bit<3>     flags;
    bit<13>    fragOffset;
    bit<8>     ttl;
    bit<8>     protocol;
    bit<16>    hdrChecksum;
    var32_t    srcAddr;
    var32_t    dstAddr;
}
```

```
struct metadata { }

struct headers {
    ethernet_t   ethernet;
    ipv4_t       ipv4;
    udp_t        udp;
}
```

Figura 12 – Cabeçalhos básicos para encaminhamento de pacotes.

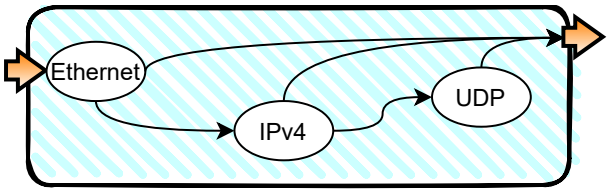


Figura 13 – Sequência de execução do *parser* básico

controle do *pipeline*, é verificando se o mesmo possui um cabeçalho IPv4 válido, sendo então aplicadas as ações definidas na tabela de *match+action*. A consulta a esta tabela é realizada com base em um algoritmo LPM (*Longest Prefix Match*[1]), utilizando o endereço IP de destino como chave da consulta, decrementando o tempo de vida do pacote e atualizando as informações de endereço físico e porta de saída, com base nos resultados da consulta, encaminhando o pacote para o *egress*, o segundo bloco de controle.

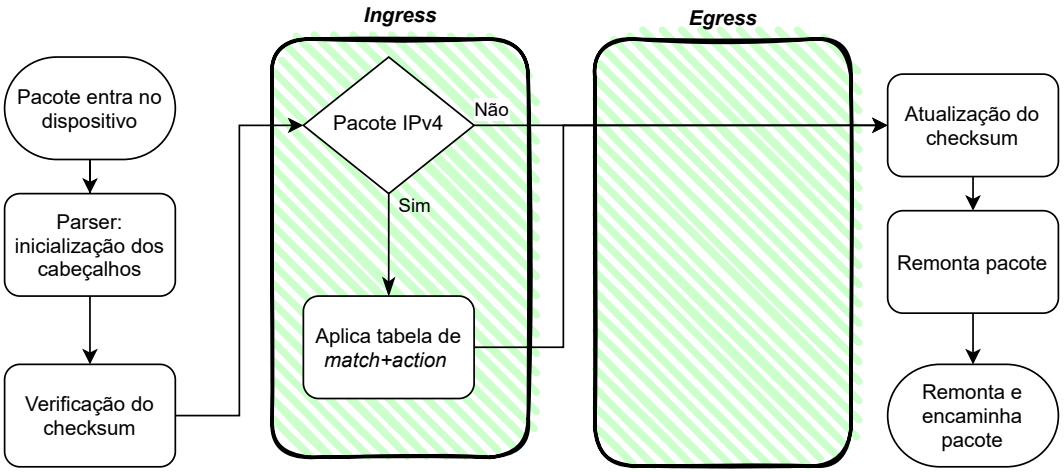


Figura 14 – Fluxo de encaminhamentos básico de pacotes.

[1] Algoritmo utilizado para selecionar uma entrada em uma tabela de encaminhamento.

O pacote pode ainda chegar diretamente ao *egress*, o que ocorre no caso de não existir um cabeçalho IPv4 válido na verificação realizada no *ingress*. Neste segundo bloco, o encaminhamento básico não realiza nenhuma computação sobre o pacote, passando diretamente para as etapas seguintes, o *compute checksum* e o *deparser*. No primeiro é realizado o cálculo do novo valor do conjunto de caracteres de validação da integridade do pacote e o segundo monta os cabeçalhos novamente no pacote para que seja então enviado para o seu destino, conforme apresentado na figura 14.

4.3 Implementação do APF

Nesta seção é descrita a abordagem proposta, tomando com base a implementação de encaminhamento básico de pacotes apresentada nas seções anteriores. Primeiramente serão abordadas as estruturas adicionais criadas e a inicialização das mesmas, seguindo com as modificações realizadas nos blocos de controle e o modulo externo utilizado. Por fim são apresentadas as limitações encontradas e a sua influência no desenvolvimento do trabalho. A codificação da aplicação na integra é apresentada no Anexo A.

4.3.1 Cabeçalhos e parser

Para a implementação do mecanismo proposto, definiram-se duas estruturas adicionais aos cabeçalhos já apresentados, denominadas *header instrinsic_metadata_t* e *header apf_t*, conforme apresentado na Figura 15. A estrutura *apf_t* é composta por quatro campos de 32 *bits* e um campo de 2 *bits*. Os campos de 32 *bits* são utilizados para armazenamento das métricas coletadas do pacote e posteriormente para permitir a coleta e análise das informações resultantes execução. O campo *analisar*, de 2 *bits*, é utilizado como variável lógica na identificação dos pacotes a serem encaminhados para o coletor ($analisar == 1$) ou não ($analisar == 0$). Por fim, este cabeçalho é adicionado a *struct headers*.

```
register<var32_t>(4) reg32;
header ethernet_t {...}
header udp_t {...}
header ipv4_t {...}

header apf_t {
    var32_t    v1;
    var32_t    v2;
    var32_t    v3;
    var32_t    v4;
    bit<2>     analisar;
}
```

```
header intrinsic_metadata_t {
    var64_t    ingress_global_tstamp;
    var64_t    current_global_tstamp;
    var32_t    janela;
    var32_t    peso_mj;
    var32_t    peso_mh;
}

struct metadata {
    intrinsic_metadata_t    intrinsic_metadata;
}
```

```
struct headers {
    ethernet_t   ethernet;
    ipv4_t       ipv4;
    udp_t        udp;
    apf_t        apf;
}
```

Figura 15 – Cabeçalho APF.

No cabeçalho *intrinsic_metadata_t* são definidos os metadados de 64 *bits* para registro de data e hora disponibilizados pela interface, e métricas de 32 *bits*. Os campos *peso_mh* e *peso_mj* permitem definir respectivamente os valores de λ e $1-\lambda$ aplicados no cálculo da m_h (Equação (4.6)). Já o campo *janela* é utilizado para definição do tamanho de J, definindo o intervalo de pacotes a ser analisado para obtenção da m_j e atualização de m_h. Estes valores são oriundos do plano de controle, apresentado no Anexo D, permitindo maior dinamicidade na configuração do ambiente.

Com relação aos registro de data e hora, embora sejam fornecidos pela definição do P4 da Netronome, necessitam ser explicitamente declarados. Nestes metadados, os 32 *bits* mais significativos representam o registro de tempo em segundos e os 32 *bits* menos significativos o registro de tempo em nanosegundos. Assim, os 32 *bits* menos significativos foram utilizados para obtenção da latência, sendo também responsável pela definição do tamanho das variáveis do cabeçalho *apf_t*, juntamente com as limitações encontradas e abordadas na seção 4.4. Por fim, este cabeçalho é adicionado a *struct metadata*.

Após a criação do cabeçalho APF, o *parser* foi incrementado para permitir a sua inicialização em todos os pacotes do tipo IPv4. Para tal, foi incluído um estado adicional entre os estados de inicialização dos cabeçalhos IPv4 e UDP, conforme apresentado na Figura 16. Esta adição comporta-se como uma extensão do estado IPv4, pois a verificação de existência de um cabeçalho UDP passa a ser realizada no estado adicionado e um pacote do tipo IPv4 não pode mais ser encaminhado ao final do *parser* sem passar pelo estado APF.

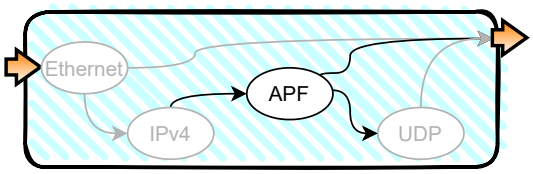


Figura 16 – Inclusão do cabeçalho APF no *parser* básico

Com relação a *struct metadata*, nenhuma alteração é necessária, visto que a mesma já é declarada por padrão nos blocos de controle. Por fim, uma estrutura vetorial de quadro posições denominada *reg*32 foi definida inicialmente com o propósito de realizar o controle da aplicação, mediante monitoramento da quantidade de pacotes e também para possibilitar a recuperação das informações externamente ao plano de dados. No entanto, conforme abordado na Seção 4.4, o uso dessa estrutura restringiu-se somente a recuperação das informações.

4.3.2 Ingress e Egress

No bloco de controle *ingress*, são criadas duas novas tabelas de *match+action*, utilizadas para definição de métricas e tratamento de pacotes anômalos. Na primeira são

definidos os pesos a serem aplicados no cálculo da m_h (Equação (4.6)), o tamanho da janela a ser computada e também variável de classificação, definindo seu valor para zero ($analisar = 0$). A segunda tabela adicionada neste bloco, corresponde a parte ao final do processamento de um pacote que deve ser encaminhando ao coletor. Como cada pacote clonado é encaminhando para o *pipeline* de entrada, é necessário diferenciá-lo de um novo pacote, o que é realizado por intermédio dos campos de metadados que definem a sua origem, permitindo especificar a procedência com interna (pacote clonado) ou externa (pacote normal) ao *pipeline* do plano de dados.

Assim, utilizando-se dessa informação, pacotes do tipo IPv4 tem então sua origem verificada. Em caso de novos pacotes, que ainda não tenham sido analisados, as tabelas de encaminhamento e definição de métricas são executadas em sequência. Já em caso de pacotes clonados, somente a tabela de monitoramento é executada. Nela é definia a porta de destino para envio do pacote ao coletor e a marcação do pacote IPv4, alterado o campo de serviço diferenciado ($diffserv = 0xC8$), permitindo também identificar estes pacotes de uma forma alternativa. Na Figura 17, são apresentadas as ações adicionais de forma destacada sobre as ações já realizadas (vide Figura 6).

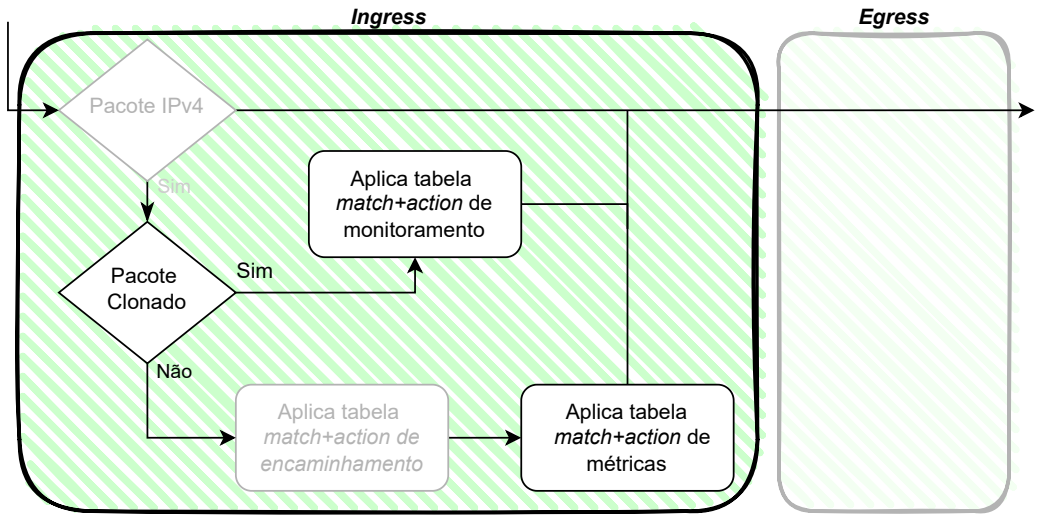


Figura 17 – Alterações realizadas no bloco de controle *ingress*.

Ao final do *ingress* o pacote é encaminhado para o bloco de controle seguinte, o *egress*, no qual são realizadas as etapas de coleta, análise e processamento das informações, conforme apresentado na Figura 18. Inicialmente é realizada uma verificação e caso seja um pacote IPv4 normal (não clonado), é realizada a recuperação das métricas de latência e tamanho do pacote, prosseguindo com a chamada do módulo externo que realiza o processamento das informações coletadas, abordado na Seção 4.3.3.

Após o processamento do módulo externo, as informações armazenadas nos campos de 32 *bits* do cabeçalho APF são salvas no registrador $reg32$. Na sequência, é realizada a verificação do campo *analisar* do cabeçalho APF, identificando os pacotes a serem encaminhados para monitoramento. Nos casos positivos, é então realizada a cópia do pacote,

a qual será redirecionada para a fila de ingresso do *pipeline*. Caso não seja necessária a análise, o cabeçalho APF é invalidado, evitando assim que os pacotes sejam perdidos. Por fim, quando os pacotes que chegam ao *egress* são identificados como clonados, é realizada somente a invalidação do cabeçalho APF, visto que o mesmo já foi processado anteriormente e seu destino já foi definido no *ingress*.

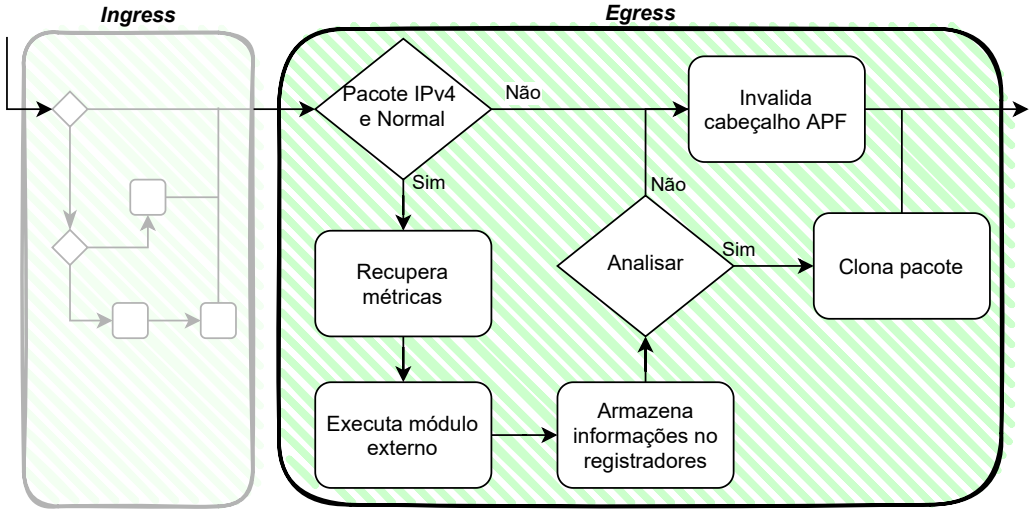


Figura 18 – Alterações realizadas no bloco de controle *egress*.

4.3.3 Módulo externo

Neste módulo ocorre o processamento das métricas coletadas resultando na média do pacote, que posteriormente é comparada com média histórica, a qual também é calculada e atualizada neste módulo, definindo assim se o pacote deve ou não ser encaminhado para monitoramento. Por fim são contabilizados os contadores utilizados para monitorar a aplicação, sendo devolvidos para o *pipeline* P4 por meio das variáveis do cabeçalho APF para armazenamento nos registradores. Ainda, para permitir o correto computo dos resultados de forma compartilhada entre as FPCs, as variáveis de controle são alocadas na memória externa EMEM.

A Figura 19, demonstra as etapas realizadas no módulo externo, apresentado no Anexo B. Primeiramente, as métricas armazenadas no cabeçalho APF são recuperas e o total de pacotes é incrementado em um variável global, nos passos (1) e (2). Neste ponto, para permitir a precisão dos resultados, é inicializado um controle de concorrência com a utilização de uma biblioteca mutex[2] disponibilizada pela Netronome, no passo (3). Após o bloqueio, é realizado o cálculo da m_p, e os valores do tamanho da janela atual (J') e da média ponderada acumulada são incrementados, nos passos (4)–(6).

Na sequência, são recuperados os valores da média histórica e do tamanho da janela atual, nos passos (7) e (8). Assim, de posse dos valores atualizados, é realizada a análise

[2] Técnica de programação concorrente utilizada para evitar que mais de um processos ou *threads* tenham acesso simultâneo a um recurso compartilhado

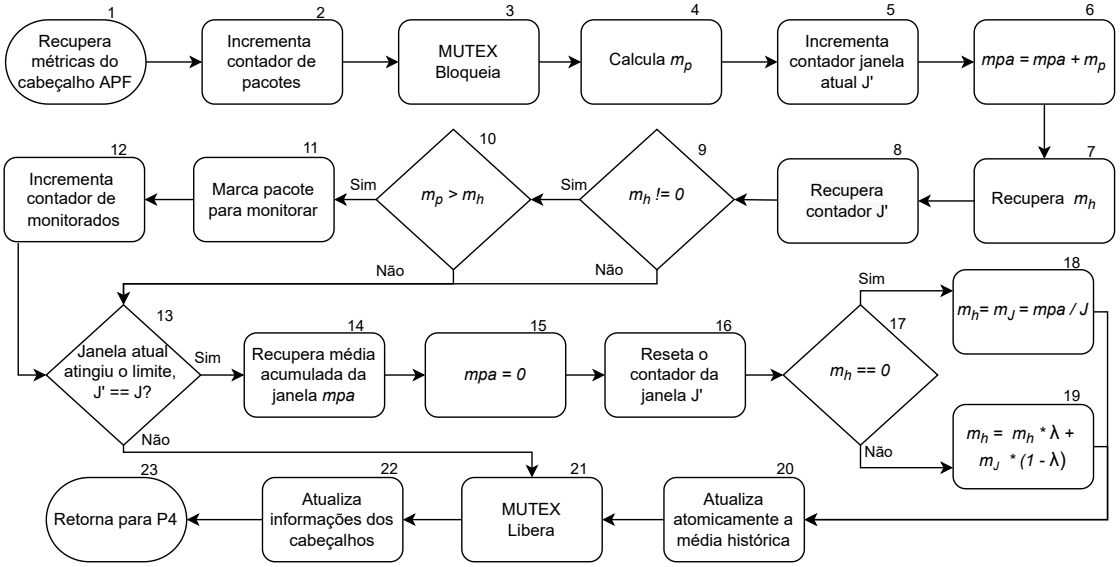


Figura 19 – Fluxo do módulo externo desenvolvido em Mircro-C.

do pacote, na qual, caso exista uma m_h já calculada e $m_p > m_h$, a variável de controle do cabeçalho APF é alterada – passos (9)–(12) respectivamente – indicando que este pacote deve ser encaminhando para o coletor. Neste ponto também é incrementado o contador de pacotes monitorados – passo (12). Após a análise do pacote, ou caso as condicionais de análise não sejam satisfeitas, é realizada a etapa de verificação e atualização da média histórica.

Nesta etapa, caso o contador de pacotes da janela atual se iguale ao intervalo pré definido, $J' == J$, o valor de *mpa* é recuperado e as variáveis J' e m_J são reiniciadas, sendo então realizada a atualização de m_h conforme descrito em 4.1, conforme as Equações (4.4) ou (4.6) – passos (13)–(19). Por fim, a nova m_h é atualizada e o bloqueio de concorrência finalizado, sendo então atualizados os campos do cabeçalho APF com as informações de quantidade de pacotes total e monitorados, bem como a m_h, permitindo a sua posterior recuperação para avaliação e monitoramento – passos (20)–(23). Todas as ações de recuperação, adição e incremento foram realizadas por funções denominadas atômicas, que possibilitam a escrita e recuperação de um valor diretamente na memória em uso, por meio do endereço de alocação desta variável, permitindo assim maior precisão nos valores obtidos.

4.4 Limitações observadas

No decorrer do desenvolvimento deste trabalho, algumas limitações existentes tanto na linguagem P4 como na arquitetura da SmartNIC resultaram em alteações na forma inicial de desenvolvimento da proposta. A limitação dos campos de metadados detalhada na Seção 4.1, exemplifica um dos casos em que foi necessário modificar a abordagem inicial, reduzindo a quantidade de métricas utilizadas. A seguir são abordadas os

demais casos e a respectiva solução adotada para casa um.

4.4.1 Operações com ponto flutuante

Um primeiro ponto foi a limitações das linguagens para realização de operações com números reais, que impactaria diretamente no cálculo das médias ponderadas utilizadas. Para contornar essa limitação foram utilizadas representações de ponto fixo para realizar as operações de multiplicação e divisão, que consiste basicamente em converter os valores em ponto flutuante (v_{real}) para inteiro (v_{int}), demonstrado na 4.7, utilizando-se de um fator escalar (SC_UP) e após realizar as operações, converter novamente o valor inteiro realizando a redução com o mesmo fator escalar utilizado 4.8.

$$v_{int} = v_{real} \cdot 2^{SC\_UP} \tag{4.7}$$

$$v_{real} = \frac{v_{int}}{2^{SC\_UP}} \tag{4.8}$$

Para definição de SC_UP, o valor recomentado equivale a metade do tamanho da variável utilizada, destinando assim os *bits* mais significativos ao valor inteiro e os *bits* restantes ao valor fracionário. Ainda, afim de evitar que o resultado da operação seja superior ao suportado pela variável, deve-se realizar um *casting*[3] de dados para um tamanho maior do que o da variável que está sendo multiplicada.

Tomando como exemplo o cálculo da média ponderada do pacote (Equação 4.9) abordada na seção 4.1, onde as variáveis utilizadas são de 32 *bits*, o fator escalar foi definido com o tamanho de 16 *bits* ($SC\_UP = 16$), sendo necessário a realização de *casting* de 64 *bits* a fim de garantir a corretude dos resultados. Dessa forma, a real operação realizada consistiu primeiramente na conversão dos fatores de multiplicação da latência (0.7) e do tamanho do pacote (0.3), para inteiros conforme demonstrado nas Equações (4.10) e (4.11) respectivamente. No entanto, estas operações foram realizadas externamente a aplicação e definidas estativamente no código, sendo as equações somente uma demonstração da operação realizada.

$$m_p = l_p \cdot 0.7 + t_p \cdot 0.3 \tag{4.9}$$

$$fator(l_p) = 0.7 \cdot 2^{SC\_UP} \tag{4.10}$$

$$fator(t_p) = 0.3 \cdot 2^{SC\_UP} \tag{4.11}$$

Após definidos os fatores de ponderação, são então obtidos os valores escalares de latência e tamanho do pacote, Equações (4.12) e (4.13) respectivamente. Também é realizado um *casting* de 64 *bits*, devido a possibilidade de um *buffer overflow*[4] resultante da

[3] Operação de conversão realizada com o objetivo de altear o tipo de uma variável

[4] Uma anomalia onde um programa, ao escrever dados em um *buffer*, ultrapassa os limites reservados e sobrescreve a memória adjacente.

multiplicação de duas variáveis de 32 *bits*, mantendo as variáveis temporárias compatíveis com o resultado esperado. O resultado real é obtido por intermédio da Equação (4.14), onde ocorre a divisão dos resultados temporários de 64 *bits* pelo valor escalar, obtendo assim o resultado correto da expressão.

$$l_{p(tmp)} = (var64\_t)fator(l_p) \cdot (var64\_t)l_pv \tag{4.12}$$

$$t_{p(tmp)} = (var64\_t)fator(t_p) \cdot (var64\_t)t_p \tag{4.13}$$

$$m_p = \frac{l_{p(tmp)}}{1 << SC\_UP} + \frac{t_{p(tmp)}}{1 << SC\_UP} \tag{4.14}$$

Ainda, visto que a operação de potenciação também não é diretamente suportada pela linguagem P4, a sua utilização foi contornada com a realização da operação de deslocamento de *bits*, onde a operação de deslocamento para esquerda $a << b$ equivale a operação de multiplicação $a \cdot 2^b$. Dessa forma a operação $1 << SC\_UP$ utilizada na Equação (4.14) equivale à $1 \cdot 2^{SC\_UP}$, realizando a divisão pelo valor necessário.

4.4.2 Limitações da interface

Além as limitações na quantidade de metadados, a interface da Netronome apresentou outras limitações conforme descrito a seguir:

- **operações:** adicionalmente ao fato de não serem aplicadas à números reais, as operações de multiplicação e divisão também são limitadas a variáveis de no máximo 32 *bits* (VIEGAS et al., 2021b), corroborando para definição das variáveis do cabeçalho APF;
- **deslocamento de *bits*:** a realização destas operação não permite a utilização de variáveis definidas em tempo de execução, sendo realizadas somente com valores definidos em tempo de compilação, valores estáticos. Esta limitação impacta diretamente na configurabilidade e dinamicidade da aplicação, tornando-a mais limitada nestes quesitos (VIEGAS et al., 2021b);
- **registradores:** durante o desenvolvimento da proposta verificou-se a impossibilidade de utilização dos registradores nas tarefas de contabilidade e recuperação das quantidades dos pacote processados e monitorados. Nos experimentos realizados, foi percebido que os valores armazenados nos registradores foram sempre inferiores aos contadores disponibilizados pela interface e também pelo registro de pacotes enviados e recebidos pelo gerador de tráfego, os quais se equivalem. Assim, esta limitação colaborou fortemente para adoção do módulo externo em Micro-C.
- **estruturas de controle:** no decorrer da implementação foi verificada também a impossibilidade da utilização de valores recuperados dos registradores na comparação em estruturas de controle, sendo esse também um dos principais motivos para

adoção do módulo externo, visto que o controle do tamanho do intervalo J é uma ação indispensável para o desenvolvimento do trabalho.

4.4.3 Transbordo da média acumulada

Inicialmente o somatório da média ponderada acumulada foi realizado com os valores brutos das métricas coletadas. No entanto, conforme pode ser observado nas três últimas linhas da Tabela 3, tais valores se mantém nas casas de dezenas e centenas de milhares chegando até a casa dos bilhões. Considerando que na obtenção da *mpa*, ocorre ainda computação de uma segunda métrica, estes resultados acabaram gerando valores muito elevados.

Nos cálculos realizados, foi inicialmente considerada apenas uma porção da latência para composição da métrica, sendo o tamanho dos pacotes, expressivamente inferior. Ainda assim, o resultado de *mpa* se mostrou muito superior aos valores suportados pelas variáveis de 32 *bits*, principalmente à medida que o intervalo de janela se elevava. Apenas intervalos de até no máximo aproximadamente 32 mil pacotes seriam possíveis de analisar, limitando drasticamente a análise almejada, devido ao curto intervalo com relação a tráfego suportado.

Uma alternativa para janela maiores seria a utilização de variáveis de 64 *bits*, o que não se demonstrou viável devido a sua estrutura. Da mesma forma que os metadados de *timestamp*, as variáveis de 64 *bits* são formadas por duas palavras de 32 *bits*. Assim, nos testes realizados, os valores armazenados nas variáveis maiores ocupam sempre os 32 *bits* mais significativos, gerando valores incorretos e elevados em relação ao resultado esperado. Ainda como forma de contornar esta limitação, foram realizados testes com a operação de deslocamento de *bits*. Neste caso, embora a operação tenha ocorrido conforme esperado, o resultado foi aplicado somente nos *bits* mais significativos, não se mostrando como solução para o problema.

Assim, para contornar esta limitação, foi então aplicada uma equivalência de operações. As médias ponderadas foram divididas pela maior constante disponível, que não implica-se em valores nulos para as métricas mais baixas. A adoção desta solução garantiu primeiramente o contorno da limitação, gerando resultados corretos para o cálculo da média utilizadas. Adicionalmente, permitiu também a realização testes com janelas maiores, na casa das centenas de milhões de pacotes.

4.4.4 Fluxo de pacotes

O processamento paralelo e multi core presente na arquitetura das SmartNICs (vide Seção 2.3.1) proporciona um elevado desempenho, alcançando a casa de dezenas de milhões de pacotes por segundo. No entanto, no decorrer do desenvolvimento tais características acabaram afetando a exatidão dos dados computados. Inicialmente foram aplicadas configurações de compartilhamento de memória e acesso global por todas as

ilhas de processamento. Porém, tais configurações não foram suficientes para garantir a corretude das operações realizadas.

A correta computação das médias necessárias para o desenvolvimento deste trabalho foi então obtida com a limitação da quantidade de ilhas de FPCs e CPUs utilizadas no processamento. Tais recursos foram limitados, reduzindo a quantidade de ilhas de FPCs de 5 para 1 e a quantidade de FPCs de 12 para 5. Tal limitação resultou diretamente no desempenho conforme pode ser observado na Tabela 3, onde a coluna (A) demonstra os resultados de uma execução realizada sem limitação nos recursos da interface e a coluna (B) apresenta os resultados obtidos com as limitações supracitadas.

Tabela 3 – Resultados da limitação de processamento em 105s de execução.

Execução	A	B
Total de pacotes	1.564.286.147	135.761.377
Pacotes por segundo	14.548.077	1.292.965
Latência Mínima	2.721 ns	72.332 ns
Latência Média	5.616 ns	374.719 ns
Latência Máxima	54.466 ns	3.295.522.604 ns

Mediante os resultados apresentados é possível visualizar que o impacto não se da somente na quantidade de pacotes mas também na latência, neste caso calculada pela diferença entre o *timestamp* de entrada e o *timestamp* atual do pacote, onde a diferença se torna muito mais expressiva. Tais limitações ainda não contabilizam a utilização do controle de concorrência abordado na seção 4.3.3, o qual por sua vez também adiciona uma sobrecarga sobre o processamento, conforme abordado em Santos e Luizelli (2021).

4.4.4.1 Contorno da limitação de processamento

O uso de mecanismos como mutex e semáforo[5], para garantir o acesso único a variáveis globais, não se mostrou eficaz em um primeiro momento. Devido a arquitetura da interface e a forma de implementação, tais mecanismos não garantiram a realização correta dos cálculos efetuados. As variáveis de controle dos mecanismos, necessitam ser armazenadas em uma memória de acesso comum a todas as ilhas (EMEM ou IMEM). Porem, devido ao paralelismo implementado neste acesso e a execução de um instância independente do código em cada ilha de processamento, a exclusão mútua com uma estrutura única (Figura 20(a)) não era garantida.

No entanto, a solução apresentada por Vestin et al. (2021), busca contornar este problema adotando a utilização de um mecanismo de controle vetorial. Utilizando uma

[5] Conceito criado por Dijkstra para solucionar conflitos de acessos concorrentes a um mesmo recurso por processos distintos.

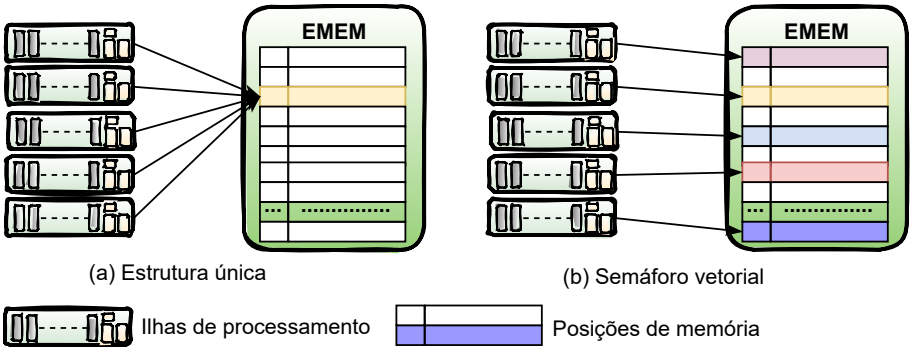


Figura 20 – Acesso a memória com os controles de concorrência adotados.

implementação do mecanismo semáforo, similar a descrita em Wray (2014), são realizados ajustes para o funcionamento como estrutura vetorial. Em cada pacote, um campo adicionado ao cabeçalho serve como identificador do fluxo. Este campo é então utilizado como uma chave para identificar qual posição do semáforo deve ser utilizada para realizar o bloqueio. Dessa forma, a limitação é contornada com a utilização de mecanismos por fluxo, e não por de um mecanismo central para todos os pacotes.

Porém, visto que buscava-se não realizar alterações nos pacotes, e que o trabalho de Vestin et al. (2021) não considera variáveis globais, como a mh e a mpa aqui utilizadas, algumas adaptações foram realizadas. Inicialmente, foi implementada uma função *hash* de modo a ser utilizada como indexador do vetor de semáforos. Esta configuração se torna viável, visto que cada pacote é direcionado para uma ilha de processamento de acordo com o as informações de seu cabeçalho (vide seção 2.3.1). Assim, foram utilizados os endereços IP de origem e destino para obtenção do indexador do vetor de semáforos. Com isso, cada conjunto de pacotes acessaria somente uma posição de memória, tratando a concorrência e o acesso indevido, conforme demonstrado na 20(b).

A utilização de um mecanismo de controle vetorial altera de certa forma o funcionamento e os resultados obtidos com a utilização de uma única ilha. Com a separação dos pacotes processados, se tornou necessário também contabilização das métricas por fluxo para complementar o correto controle de concorrência. Assim, as métricas de média ponderada acumulada (mpa), média histórica (mh)e contador de janela J, foram também vetorizados. Ou seja, cada conjunto de fluxos identificado por um mesmo indexador – ou cada ilha de processamento – possui este conjunto de métricas e contadores próprios. Dessa forma, adotando o controle de concorrência vetorial com semáforos, apresentado no Anexo C, se torna possível alcançar um melhor desempenho de modo geral, conforme apresentado na seção 5.2.2.

5 AVALIAÇÃO

Neste capítulo é apresentada a avaliação do trabalho desenvolvido. Inicialmente são abordados o ambiente de desenvolvimento e a definição do caso de uso adotado. Em seguida, são apresentadas a metodologia e os cenários de testes utilizados para obtenção dos resultados. Por fim, são demonstrados os resultados obtidos com processamento limitado (1 ilha de processamento) e sem limitação (5 ilhas de processamento), sendo realizada uma análise sobre os mesmos.

5.1 Ambiente e metodologia adotados

5.1.1 Ambiente de desenvolvimento

O ambiente de desenvolvimento é composto por três servidores com processador AMD Ryzen 7 3800X, com 8 cores e 24GB de RAM, utilizando o sistema operacional Ubuntu 16.04.7 LTS. Dois destes servidores foram utilizados como dispositivos geradores de tráfego (d_A e d_B), contando cada um com uma SmartNIC Agilio CX 2x10GbE, composta por duas interfaces de rede SFP+. O terceiro servidor utilizado como um DUT (*Device Under Test*), sendo composto por uma SmartNIC Agilio CX 2x40GbE, composta por duas interfaces de rede QSFP+. Assim, a aplicação desenvolvida é carregada e executada no DUT, que atua como um dispositivo de encaminhamento (d_C).

Para interconexão entre as interfaces de rede SFP+ e QSFP, foi aplicada uma configuração sobre a Agilio CX 2x40GbE, subdividindo a mesma em 4 interfaces virtuais de 10GbE, accessíveis por meio de um cabo *breakout*[1] QSFP+ para SFP+. Dessa forma, cada interface de 40GbE da SmarNIC presente no DUT, foi segmentada em quatro interfaces de 10 GbE, compatíveis com as interfaces SFP+, resultado no ambiente de execução demonstrado na Figura 21.

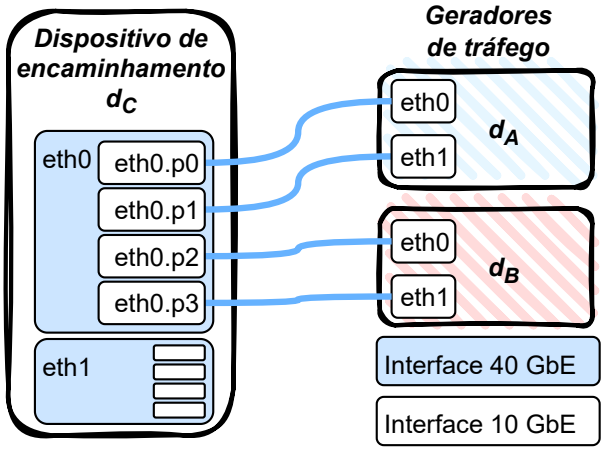


Figura 21 – Ambiente de desenvolvimento.

[1] Modelo de cabo que permite compartilhar uma porta, convertendo uma conexão 1:1 em 1:N.

Como gerador de tráfego nos dispositivos d_A e d_B, foi utilizado o DPDK (*Data Plane Development Kit*[2]) disponibilizado pelo MoonGen (EMMERICH et al., 2014). As execuções foram realizadas com a biblioteca Netronome Packet Generator, disponibilizada pela Netronome diretamente no MoonGen. Foram realizados o envio de pacotes IPv4 de 64B tendo como origem, endereços IP dinâmicos, e como destino, endereços IP fixos.

5.1.2 Ataque de negação de serviço como estudo de caso

Como forma de demonstrar o funcionamento da aplicação desenvolvida para fins didáticos, foi adotado como caso de uso, o cenário de ataque de negação de serviço (*Denial of Service* - DoS). Tais ataques visam dificultar ou impedir o acesso a redes de computadores, servidores ou sistemas *online*, entre outros. A sua realização pode ocorrer a partir da exploração de vulnerabilidades ou do esgotamento de recursos. No primeiro caso, são exploradas falhas em aplicações, sistemas operacionais ou protocolos, por exemplo. Em alguns casos, um ataque por vulnerabilidade por ser realizado com somente um pacote contendo as características que ativam a vulnerabilidade.

No segundo caso, a disponibilidade do serviço fica comprometida pelo consumo de recursos essenciais necessários ao seu funcionamento, como memória, processamento e largura de banda da rede. A limitação destes recursos faz com que o envio de um grande número de mensagens acabe por saturar a sua capacidade. Dessa forma, com o esgotamento dor recursos, o alvo do ataque não consegue prestar o serviço necessários. Ainda, os ataques DoS podem ser realizados de forma distribuída e coordenada, fazendo uso de um conjunto de computadores, este ataque é denominado de *Distributed Denial of Service* (DDoS) (LAUFER et al., 2005) .

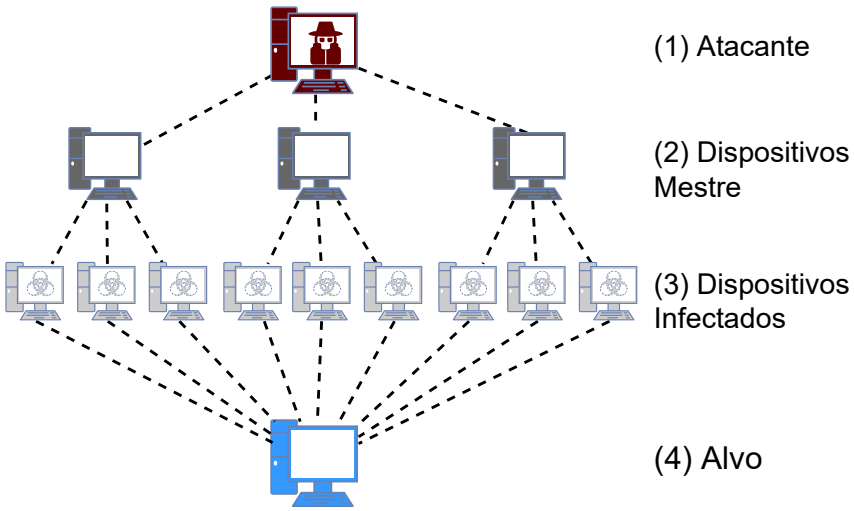


Figura 22 – Ataque distribuído de negação de serviços.

Em um ataque distribuído, demonstrado na Figura 22, um atacante (1), coor-

[2] https://www.dpdk.org/

dena um o mais computadores mestres (2). Por meio de um *malware*[3], os computadores mestre detém o controle de outros dispositivo. Sob o controle dos dispositivos mestre, os equipamentos infectados (3), passam a direcionar seu tráfego para um único alvo (4) em um mesmo instante de tempo. Dessa forma, a grande quantidade de equipamentos realizando requisições ao dispositivo alvo de forma simultânea, causa um esgotamento nos recursos disponíveis, permitindo que o atacante alcance seu objetivo. Na seção seguinte são detalhadas as configurações utilizadas para simular um cenários de DDoS.

5.1.3 Metodologia e configurações

A metodologia utilizada na realização dos experimentos foi baseada em intervalos de execução, sendo demonstrada de forma geral na Figura 23. A partir do dispositivo d_A, um trafego de rede é enviado de forma constante durante todo o tempo de execução do experimento (t_T). Simultaneamente, o dispositivo d_B encaminha rajadas de tráfego são enviadas de forma intercalada, buscando simular cenários de congestionamento de rede por negação de serviço. Ainda, os intervalos utilizados ($t_1, t_2 ... t_n$) possuem a mesma duração, de modo que o extensão de um ataque seja a mesma de um intervalo sem ataque. Por fim, a aplicação desenvolvida é descarregada no dispositivo d_C, o qual desempenha simultaneamente a função de analisar e coletar as informações INT.

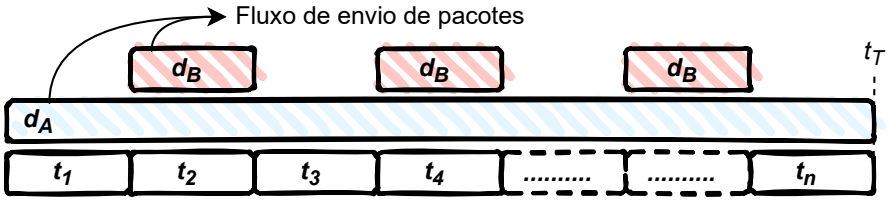


Figura 23 – Metodologia de geração de tráfego aplicada.

Os resultados foram obtidos mediante a adoção de uma média simples de um conjunto de 10 execuções sequenciais, devido a confiabilidade obtida, sendo os valores agregados, apresentados juntamente com o Desvio Padrão (DP)[4]. Para realização dos experimentos, foi adotado um tempo total de execução de 210 segundos, pelo qual o fluxo contínuo é executado. Ainda, seguindo a metodologia definida, foram estabelecidos 4 cenários, nos quais são realizadas variações na duração e consequentemente na quantidade dos intervalos. Para cada um dos cenários adotados, foram efetuados dois conjuntos de testes, realizando variações de parâmetros como peso e tamanho de janela. A seguir são descritos os cenários utilizados, bem como as configurações utilizadas nos conjuntos de testes.

[3] *Software* malicioso projetado para se infiltrar em um dispositivo, permitindo a realização de ações sem o conhecimento do usuário.

[4] Parâmetro que indica o grau de variação de um conjunto de elementos, obtido por $\sigma = \sqrt{\sum \frac{(x_i - \bar{x})^2}{n}}$.

- **Cenário 1**: são utilizados intervalos de 16 segundos, resultado em 13 intervalos. Conforme apresentado na Figura 24(a), são simulados 6 ataques, realizados nos intervalos t2 (16s–32s), t4 (48s–64s), t6 (80s–96s), t8 (112s–128s), t10 (144s–160s) e t12 (176s–192s);

- **Cenário 2**: são utilizados intervalos de 30 segundos, resultado em 7 intervalos. Conforme apresentado na Figura 24(b), são simulados 3 ataques, realizados nos intervalos t2 (30s–60s), t4 (90s–120s) e t6 (150s–180s);

- **Cenário 3**: são utilizados intervalos de 42 segundos, resultado em 5 intervalos. Conforme apresentado na Figura 24(c), são simulados 2 ataques, realizados nos intervalos t2 (42s–84s) e t4 (126s–168s);

- **Cenário 4**: são utilizados intervalos de 70 segundos, resultado em 3 intervalos. Conforme apresentado na Figura 24(d), é simulado 1 ataque, realizado no intervalo t2 (70s–140s);

- **Variação de janela**: nesta configurações foram realizadas variações nos intervalos de pacotes, com valores de J entre 2^{20} e 2^{23}, utilizando-se de um multiplicador de ponderação elevado ($\lambda = 0.9$), representando assim uma maior importância para a média histórica;

- **Variação do λ**: nesta configuração foi utilizando somente $J = 2^{20}$, realizando a variação de λ entre 0.1 e 0.9, buscando analisar o impacto da variação do peso aplicado as métricas utilizadas;

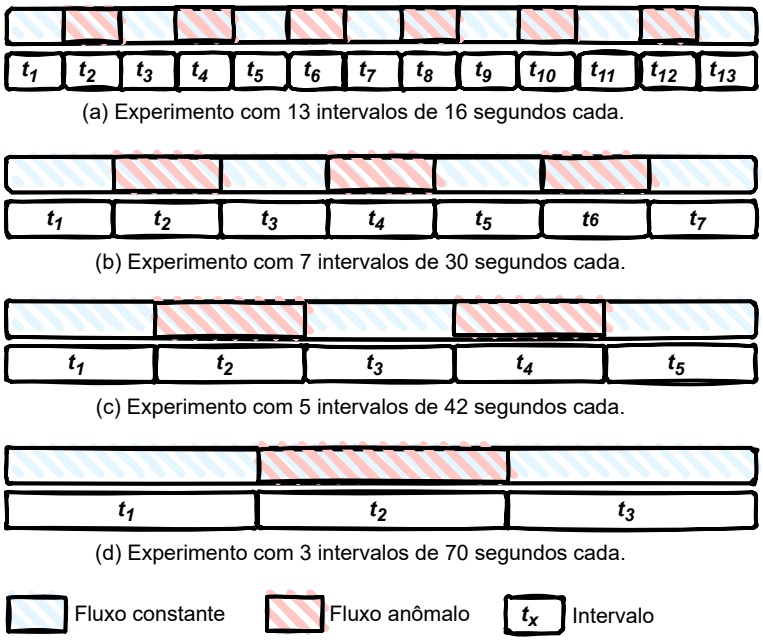


Figura 24 – Cenários de execução utilizados.

Ainda, tendo em vista que dentre os trabalhos relacionados não foram identificadas aplicações que realizem a mesma implementação, foram realizados mais dois conjuntos de execuções. O primeiro com a implementação de uma metodologia clássica de monitoramento INT onde todos os pacotes são encaminhados para o coletor. O segundo caso consiste em uma aplicação que realiza a comparação com base em uma média estática, oriunda da análise de comportamento do pacotes encaminhados por um único fluxo constante entre d_a e d_c.

Para obtenção da média estática, foram realizadas 10 execuções com um intervalo de 105 segundos, cujo resultado é apresentado na Tabela 4. A partir desta distribuição, foi realizada a multiplicação da quantidade de pacotes no intervalo, pelo seu limite superior, dividindo o resultado pelo valor total de pacotes. Neste cálculo, foram desconsiderados os resultados superiores a 512.000, devido a grande amplitude dos valores de média ponderada e a baixa distribuição de pacotes neste intervalo.

Tabela 4 – Distribuição de pacotes por média ponderada.

Média ponderada	**Quantidade de pacotes**	**Porcentagem**
Maior que 512.000	39.305	0.03%
Maior que 300.000	49.185.251	42.56%
Maior que 100.000	19.880.104	17.20%
Menor que 100.000	46.458.725	40.20%
Total	115.563.385	

5.2 Resultados obtidos

5.2.1 Resultados com processamento limitado

Nesta seção são apresentados os resultados obtidos com um processamento limitado (uma ilha de FPCs). As avaliações realizadas abordam a quantidade de pacotes reportados, o desempenho e o comportamento da média histórica.

5.2.1.1 Pacotes reportados

As quantidades totais de pacotes reportados ao coletor INT em cada uma das variações de configuração e em todos os cenários avaliados é apresentada na Figura 25. A partir destes resultados, é possível observar que para o Cenário 1 (13x16s), a aplicação proposta relata em média, cerca de 78% menos pacotes quando comparada com a implementação clássica. Já quando comparada com a abordagem de média fixa, relata cerca de 25% pacotes a menos.

O mesmo desempenho pode ser observado nos cenários 2 (7x30s) e 3 (5x42s), tendo um ganho adicional de 2% no cenário 4 (3x70s). Dessa forma é possível verificar que o comportamento da aplicação não sofre alterações de acordo com a duração dos eventos. A principal razão para esse resultado, consiste no ajuste dinâmico da média histórica realizado ao longo do tempo. Ainda, se observa também o ganho de desempenho com relação as aplicações tradicionais e com média fixa, indo ao encontro do objetivo proposto.

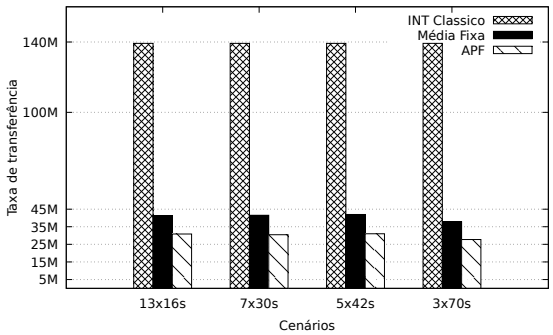


Figura 25 – Total de pacotes encaminhados ao coletor INT

De forma mais detalhada, a Figura 26 apresenta a quantidade de pacotes reportados por intervalo de tempo, para cada um dos cenários analisados. Na Figura 26(a), é possível perceber que a medida que a janela se eleva, a quantidade de pacotes monitorados tende a diminuir nos intervalos iniciais. Este comportamento ocorre justamente devido a elevação da janela, que passa a demandar um maior número de pacotes para que seja realizada a atualização da média histórica, o que acaba amenizando os valores mais desiguais. Corrobora ainda para esse comportamento a menor ocorrência da atualização da mh, também em decorrência da elevação da janela. No entanto, conforme os cenários evoluem e a duração dos intervalos se eleva, este comportamento se ameniza, devido a uma maior ocorrência de atualizações daś médias históricas.

Na Figura 26(b) se visualiza mais facilmente que nos primeiros intervalos, a quantidade de pacotes reportadas é menor que nos intervalos seguintes. Corrobora para esse comportamento, a inexistência de uma média inicial para a primeira janela de pacotes, fazendo com que não existam pacotes reportados nesse período. A baixa sobrecarga inicial existente contribui, fazendo com que a média dos pacotes neste momento não seja tão elevada. Ainda, também é possível perceber que a medida que os intervalos se elevam, a quantidade de pacotes reportados se eleva consideravelmente para cada intervalo. Este comportamento é esperado devido a quantidade de pacotes enviados ser condensada em uma quantidade menor de intervalos.

Assim, de um modo geral, é possível perceber que os comportamentos até então apresentados, ocorrem em todos os cenários, reforçando o funcionamento esperado da aplicação. Por fim, também é possível observar um comportamento dente de serra entre as barras, que representa o efeito do envio de rajadas de pacotes em um determinado

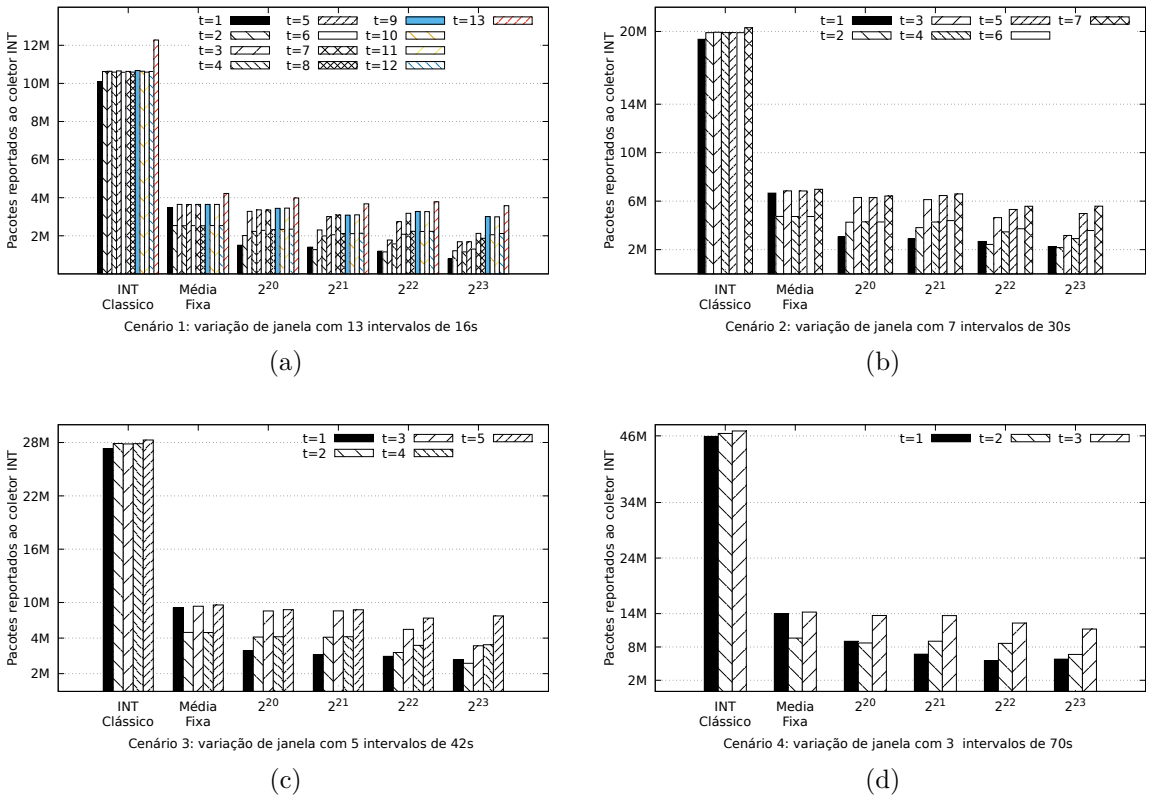


Figura 26 – Pacotes encaminhados ao coletor INT por intervalo

intervalo de tempo. Este comportamento pode ser observado nos três primeiros cenários. No Cenários 4 (Figura 26(d)), este comportamento não se mostra visível devido a baixa quantidade de intervalos (três). No entanto, apresentam o comportamento equivalente aos intervalos iniciais dos cenários anteriores, indicando que a continuidade da sua execução resultaria no comportamento equivalente.

5.2.1.2 Avaliação de desempenho

Para avaliar o desempenho da aplicação, utilizaram-se as medidas de latência e taxa de transferência de pacotes. A primeira, obtida a partir da diferença entre o *timestamp* de entrada e saída do pacote no dispositivo, expressa em nanosegundos. A segunda por meio da taxa de transferência de pacotes, expressa em pacotes por segundo (pps). A avaliação da aplicação proposta demonstra que em geral, a latência imposta é inferior as latências apresentadas pelas implementações INT Clássica e com média fixa, conforme pode ser visualizado na Figura 27(a).

De posse destes resultados, é possível perceber que a abordagem clássica e de média fixa impõem uma sobrecarga de 2,33x e 2,48x respectivamente, quando comparadas com uma aplicação de encaminhamento básico de pacotes. Já para a aplicação proposta apresenta uma sobrecarga inferior, sendo apenas 1,86x superior a latência de um encaminhamento básico. Assim, a aplicação desenvolvida se mostra mais eficiente que a

abordagem clássica e de média fixa cerca de 20% e 25% respectivamente, demonstrando o ganho de desempenho obtido com relação as demais analisadas.

Com relação a variação dos resultados, o desvio padrão obtido nos cenários de Fluxo Básico, INT Clássico e Média Fixa se mantém entre 6.96%, 9.3% e 1.24% da latência média respectivamente, demonstrando uma dispersão leve nos resultados. No entanto, para os demais cenários, estes valores passam a se elevar. A abordagem INT Clássica apresenta um DP de , se elevando para 17.51%, 21.49%, 23.76% e 22.26% para os cenários 1 à 4 respectivamente. A elevação destes valores tem relação direta com a incidência das rajadas, pois a medida que elas ocorrem, se eleva a sobrecarga e consequentemente a latência, tendo como colaborador processamento necessários para realizar o computo dos valores de cada pacote.

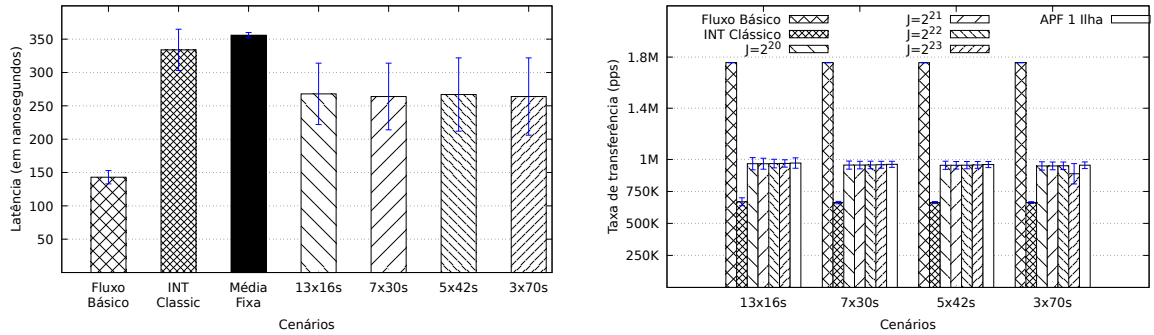


Figura 27 – Avaliação de desempenho (latência e taxa de transferências)

Na Figura 27(b), são apresentados os resultados da capacidade de encaminhamento de pacotes para todos os cenários avaliados. Inicialmente é possível aferir que em geral, a alteração das janelas em cada cenário não reflete em alterações significativas a medida que as janelas se elevam. Com relação ao desempenho, a abordagem clássica fica limitada a cerca de 37.84% do *throughput* máximo alcançado por uma aplicação de encaminhamento básico, sem sobrecarga de análise de pacotes.

Já na abordagem proposta, o volume de pacotes processados por segundo supera o volume da abordagem clássica, alcançando 54.46% do *throughput* máximo, juntamente com a abordagem de média fixa. Ainda, quando realizada uma comparação direta com a abordagem clássica, é possível aferir que a abordagem proposta apresenta um desempenho 44% superior. Com isso, é possível verificar que a aplicação proposta apresenta ganhos de desempenho, tando com relação a latência quanto com relação a taxa de encaminhamento de pacotes.

Com relação a variação dos resultados, o desvio padrão obtido com a abordagem clássica é praticamente irrelevante em todos os cenários. Nas demais execuções, esta variação se mantém estável entre os valores de 1.05% e 4.95%, com exceção do DP de 8.93% obtido para $J = 2^{20}$ no cenário 4 (3x70s). Estes valores demonstram uma dispersão

leve nos resultados, validando as análises realizadas com base na média os valores obtidos.

5.2.1.3 Variação da média histórica

Por fim, foi analisado o comportamento da média histórica para cada um dos cenários de testes adotados, avaliado em dois formatos. No primeiro formato, apresentado na Figura 28, foi realizada uma analise considerando a elevação do tamanho da janela para cada cenários, variando J entre 2^{20} e 2^{23}. No segundo, apresentado na Figura 29, adotou-se um valor fixo de janela ($J = 2^{20}$), realizando então a variação do multiplicador de ponderação λ entre 0.1 e 0.9.

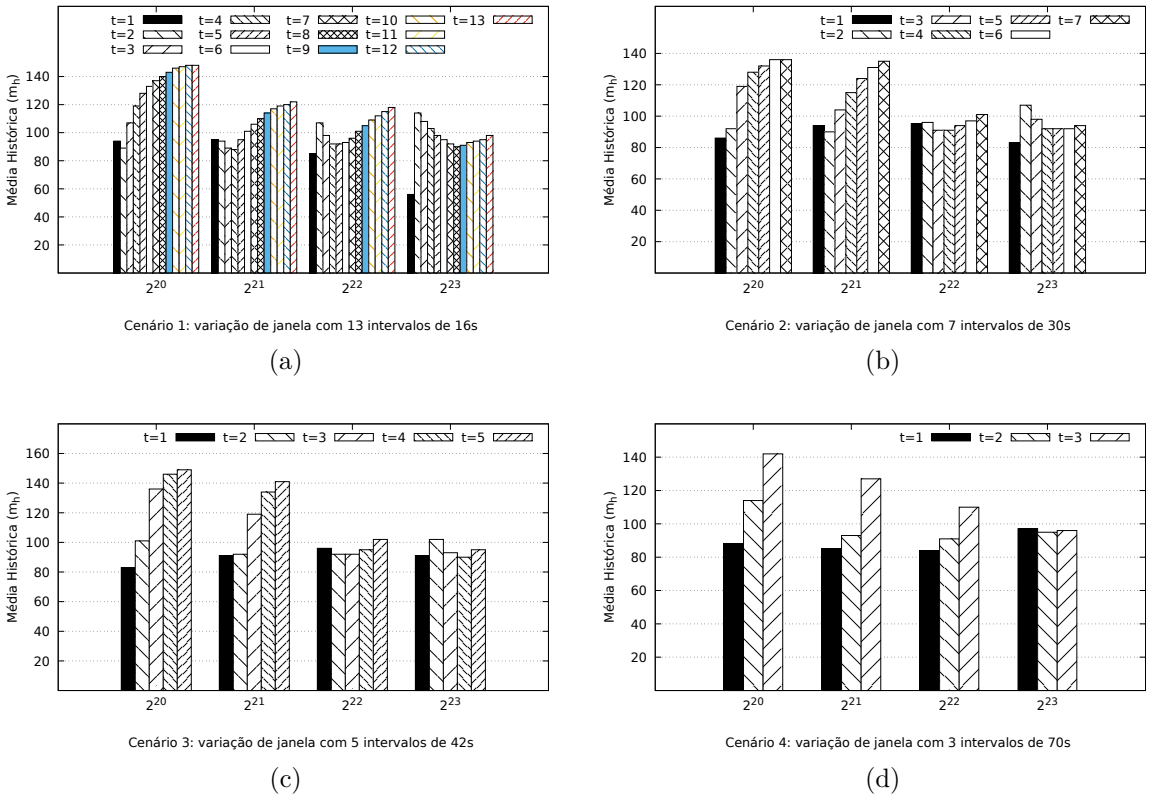


Figura 28 – Impacto da variação de J na média histórica.

No primeiro caso, é possível avaliar que para menor janela ($J = 2^{20}$), a média histórica se eleva ao longo do tempo apresentando uma tendência de estabilidade, não sendo visível somente no cenário 4 (Figura 28(d)), devido a quantidade reduzida de intervalos. Ainda, a medida que se eleva o tamanho da janela, esse comportamento tende a se inverter, apresentando uma média mais elevada nos intervalos iniciais, tendendo ainda a estabilidade, conforme visualizado no cenário 1 (Figura 28(a)).

Este comportamento vai se tornando menos evidente a medida que a quantidade de intervalos se reduz, conforme pode ser observado ao se comparar os cenários 2, 3 e 4 em sequência – Figuras 28(b),(c) e (d) respectivamente. No entanto, é possível perceber que o comportamento dos intervalos de janela apresenta um funcionamento equivalente em

todos os cenários. Essa alteração no comportamento da média a medida que se elevam as janelas, ocorre devido ao fato de a somas iniciais das janelas maiores (2^{22} e 2^{23}) englobarem as primeiras rajadas enviadas.

Ao avaliar o primeiro intervalo de cada janela, se verifica que ocorre uma diminuição do valor computado, a medida que J se eleva, a mh diminui. No entanto, a medida que os cenários evoluem, as médias vão se elevando, chegando ao cenário 4 com um resultado mais próximo aos intervalos seguintes. Conforme já abordado, tal comportamento deriva da menor ocorrência de atualização da mh em janelas mais elevadas nos cenários iniciais, que passam a ter maior incidência a medida que os cenários evoluem, devido a elevação dos intervalos.

Ao analisarmos a variação do λ entre 0.1 e 0.9, conforme apresentado na Figura 29(a), é possível observar que valores mais baixos priorizam o comportamento recente. Isso faz com que as rajadas de tráfego alterem as métricas do plano de dados (por exemplo, tempo de processamento na rede) intensificando a propagação destas métricas para as janelas seguintes. Já para os valores mais elevados, tendem a manter uma média mais estável, devido a maior importância atribuída a média histórica no momento da atualização. Dessa forma, valores mais elevados tendem a priorizar um comportamento histórico, sendo menos suscetíveis a alterações recentes.

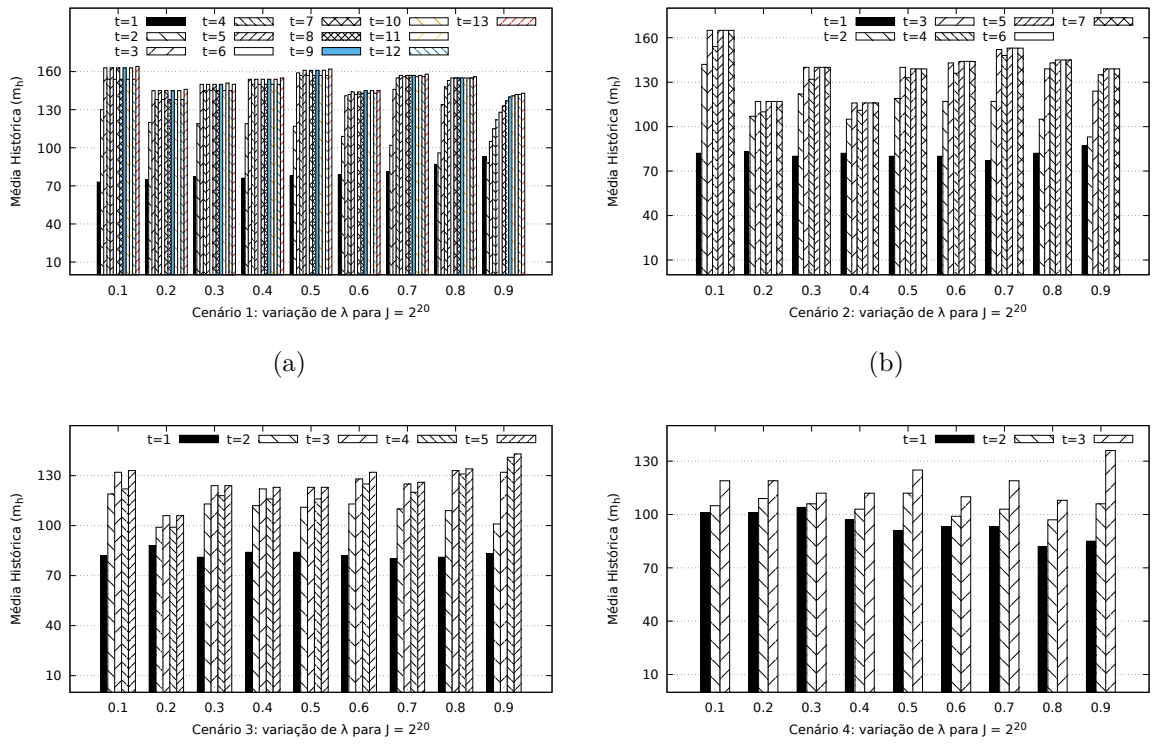


Figura 29 – Impacto da variação de e λ média histórica em $J = 2^{20}$.

Ao se comparar todos os cenários, é possível perceber que este comportamento

não sofre alteração. No entanto, a medida que se reduzem os intervalos de execução, esta avaliação tende a ser menos evidente. Por exemplo, no cenário 4 (Figura 29(d)), devido ao pequeno número de intervalos, já não se visualiza de forma clara o comportamento descrito. No entanto, assim como na avaliação da variação de janela, é possível observar a correspondência de comportamento entre os intervalos em todos os cenários. Dessa forma, é possível verificar que a aplicação apresenta não sofre alterações comportamentais a medida que se elevam a duração dos intervalos ou se diminuem as suas quantidades.

5.2.2 Resultados sem limitação de processamento

Ao contornarem-se as restrições da utilização das múltiplas ilhas de FPC, foram realizadas novas baterias de testes para validar funcionamento da aplicação. Assim, nesta seção são apresentados os resultados obtidos com a elevação do poder de processamento. As avaliações realizadas abordam a quantidade de pacotes reportados e o desempenho da aplicação, realizando também a comparação com os resultados anteriores.

Nos novos testes realizados, a abordagem com média estática não foi incluída devido aos requisitos para sua definição. Ao se adotar uma execução por fluxos, a média estática deveria ser obtida em cada fluxo, podendo compor uma única média ou uma média para cada processador. Ainda, a sua inexpressividade para os objetivos dos testes realizados nesta etapa, somados a questões como restrições da interface, desenvolvimento, validação e execução, corroboraram para não adoção desta abordagem.

5.2.2.1 Pacotes reportados

A Figura 30, apresenta a quantidade de pacotes reportados por intervalo de tempo para cada um dos cenários analisados. Em comparação com a abordagem clássica, é possível observar que os cenários analisados reportam em média 63% menos pacotes. Este valor se mantém próximo aos 78% apresentados no processamento limitado (seção 5.2.1.1). Também é possível observar o mesmo comportamento apresentado com os resultados de processamento limitado, onde, no primeiro intervalo, a quantidade de pacotes encaminhados ao coletor é inferior e tende a diminuir a medida que as janelas se elevam.

Mantendo ainda o funcionamento esperado, os gráficos apresentam o mesmo comportamento dente de serra já demonstrado. No entanto, com o processamento limitado, os picos de envio ocorreram no intervalo posterior a rajada. Já sem esta limitação, estes picos passaram a ocorrer dentro do mesmo intervalo onde ocorrem as rajadas. Este comportamento demonstra que a elevação do poder de processamento permite uma resposta ainda mais eficaz, fazendo com que as anomalias sejam detectadas tão logo ocorram.

Estes resultados corroboram com o funcionamento proposto da aplicação, visto que o seu comportamento não se mantém somente na variação dos cenários, mas também com a elevação do poder de processamento. Demonstram também que quanto maior o

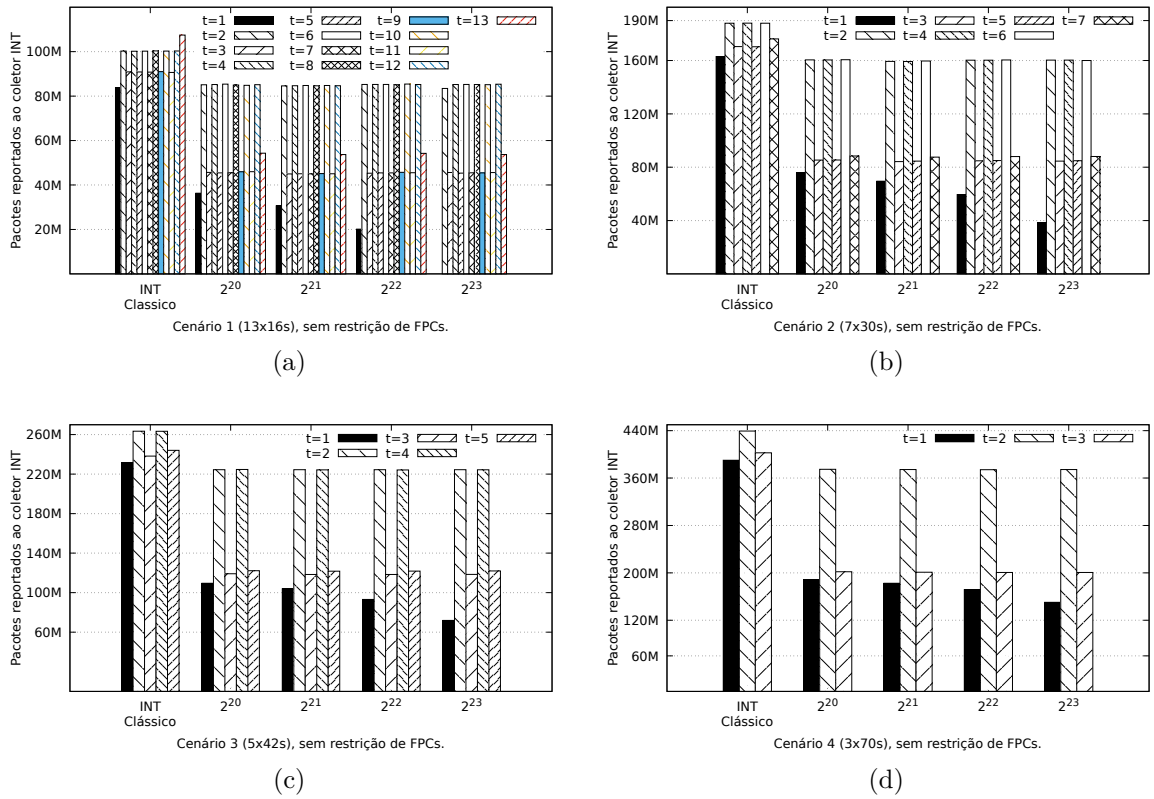


Figura 30 – Pacotes encaminhados ao coletor INT por intervalo sem restrição de FPCs

poder de processamento, melhor o desempenho da aplicação com relação a detecção de anomalias.

5.2.2.2 Avaliação de desempenho

A avaliação de desempenho realizada aborda a latência e taxa de transferência de pacotes. Inicialmente, na Figura 31, são apresentadas as taxas de transferência de pacotes para a abordagem INT Clássica e variação de janelas, para cada cenário. Nestes resultados é possível avaliar um comportamento similar em todas as variações de janela, onde a quantidade de pacotes processados se eleva nos intervalos de ocorrência das rajadas. Ainda se visualiza que esta capacidade se acentua no primeiro intervalo, sendo mais visível nos dois primeiros cenários, Figura 31(a) e (b). Este comportamento ocorre pela ausência de uma média histórica inicial, que reduz a sobrecarga no processamento e pela elevação do tamanho da janela, que retarda a obtenção desta primeira m_h.

Também é possível visualizar um sutil ganho de desempenho em relação a abordagem INT Clássica. Esse ganho ocorre em todos os intervalos das janelas, e varia entre 1,57% e 1,37% para os Cenários 1 e 4 respectivamente. Na Figura 32(a), os resultados da capacidade de encaminhamento são apresentados de forma agregada, juntamente com o resultado de encaminhamento básico e da aplicação com processamento limitado. Ao avaliar-se estas informações, é possível verificar que tanto a aplicação desenvolvida como

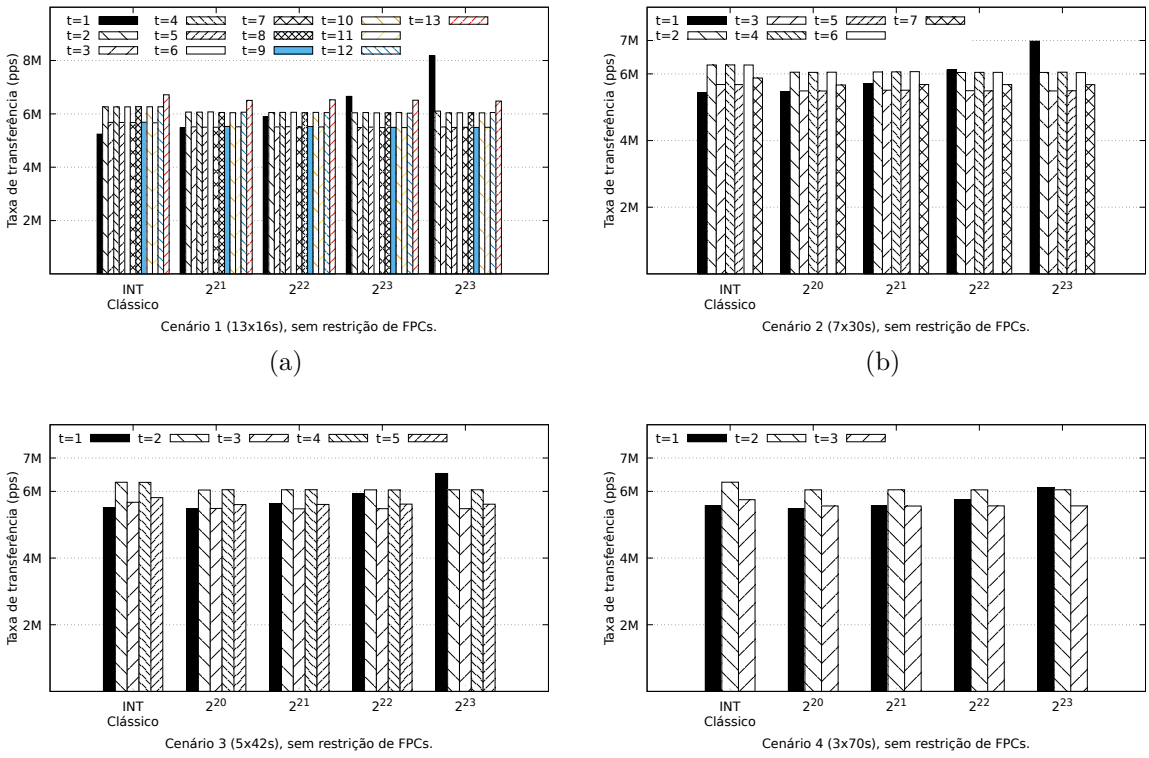


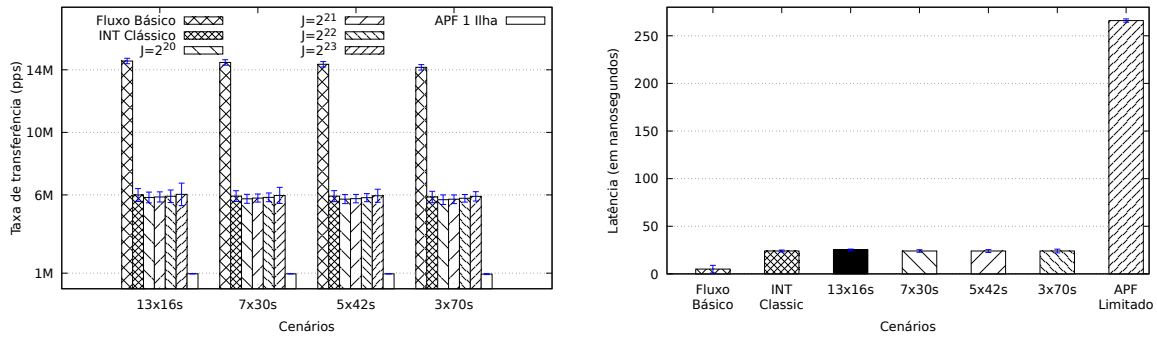
Figura 31 – Taxa de transferência por cenários.

a abordagem clássica alcançam no máximo, cerda de 41% do *throughput* obtido por uma aplicação de encaminhamento básico. Este resultado fica próximo do desempenho de 54.46% obtido com a limitação de processamento (seção 5.2.1.2).

No entanto, essa diminuição não pode ser analisada separadamente, mas sim em conjunto com o ganho de desempenho obtido ao se utilizar todas as ilhas de FPCs disponíveis. Ao se comparar os processamentos realizados com 1 e com 5 ilhas, é possível aferir um ganho de 6,06x na taxa de transferência. No primeiro caso, a capacidade de encaminhamento máxima obtida foi de 1.081.691pps, contra 8.176.195pps obtidos no segundo caso, respectivamente no primeiro e último intervalo de uma janela $J = 2^{23}$ do cenário 1. Assim, de um modo geral, é possível observar também que a melhora de desempenho não altera o comportamento da aplicação em ambos os casos.

Ainda com base nos resultados demonstrados na Figura 32(a), é possível verificar que a variação dos resultados apresenta uma leve distorção. Numericamente, este valores se mantém entre 4.19% e 8.6% para todos os resultados, com exceção dos 11.93% obtidos na execução da janela $J = 2^{23}$ no cenário 2 (7x30s). Dessa forma, os valores obtidos demonstram a constância nas execuções corroborando com as avaliações realizadas.

Na Figura 32(b), são apresentadas as medidas de latência com 5 ilhas de processamento e a média a aplicação com 1 ilha. Nos resultados apresentados, é possível verificar que tanto a aplicação desenvolvida como a abordagem clássica apresentam praticamente



(a) (b)

Figura 32 – Avaliação de desempenho sem limitação de FPCs

o mesmo desempenho. Dessa forma, a sobrecarga imposta por ambas as abordagens é ceca de 4.90x quando comparada com o encaminhamento básico de pacotes. No entanto, quando comparada com a abordagem limitada, o ganho de desempenho é de 10,87x.

A obtenção de desempenhos similares, tanto para capacidade de encaminhamento como taxas de latência, demonstra que a aplicação não gera uma sobrecarga adicional. Ainda, é possível inferir que este comportamento se dá devido a elevação de poder de processamento. Com este ganho de desempenho, a abordagem clássica consegue se equiparar a abordagem proposta, ao contrário do que é observado na analise de desempenho com limitações (seção 5.2.1.2).

Com relação a variação dos resultados, o desvio padrão da média das latência para o Fluxo Básico de execução é igual a 4. Este valor pode ser elevado levando em consideração a latência média obtida. No entanto, a grande variação se encontra exatamente nos intervalos onde ocorrem as rajadas, fazendo com que ocorra uma sobrecarga no processamento e dessa forma de tenham valores mais dispersos de latência. Com relação aos demais cenários 1 à 4, o valore se mantêm entre 1.06 e 1.95 e para a abordagem Clássica o valor é de 1, demonstrando a constância nas execuções corroborando com as avaliações realizadas.

6 CONSIDERAÇÕES FINAIS

Neste capítulo são inicialmente apresentadas as conclusões obtidas neste trabalho. Na sequência, são apresentados algumas sugestões de trabalhos futuros. Por fim, apresenta-se a relação das publicações diretamente relacionadas a esta dissertação.

6.1 Conclusão

A necessidade de monitoramento dos ambientes de rede é uma questão indiscutível para manter a disponibilidade dos serviços, sendo também imprescindível que métodos mais eficazes sejam implementados de acordo com o avenço tecnológico. Nesse sentido, o uso de INT com abordagem de monitoramento de rede permite alcançar tais objetivos, proporcionando maior visibilidade de uma forma mais refinada. No entanto, por se tratar de uma tecnologia emergente, muitas questões ainda se encontram em aberto para que sejam possível realizar uma utilização ampla e eficaz dessa metodologia de monitoramento.

Dentre estas pendências, a orquestração dos dados a serem encaminhados para um coletor e as rígidas restrições apresentadas pelas interfaces de rede inteligentes são problemas que embora tenham sido abordados na literatura atual, ainda são questões em aberto. Assim, a proposta e implementação de um mecanismo leve baseado em Média Móvel Exponencialmente Ponderada dentro do plano de dados SmartNIC para auxiliar o processo de tomada de decisão de reportar dados INT, se mostra como uma proposta que vai ao encontro das necessidades atualmente em aberto nesta linha de pesquisa.

Assim, este trabalho apresenta como contribuições: *(i)* a implementação de um mecanismo seletivo para encaminhamento de informações de INT diretamente no plano de dados, *(ii)* a verificação das limitações existentes em dispositivos de rede programáveis, no que tange ao projeta e implementar operações mais complexas, *(iii)* o desenvolvimento e apresentação de propostas para contornar as restrições existentes na arquitetura de SmartNICs levantadas durante o desenvolvimento da aplicação *(iv)* a demonstração de viabilidade e funcionamento por meio da apresentação dos resultados obtidos, alcançando assim as contribuições propostas.

Com relação aos resultados obtidos, a realização da análise de pacotes de forma dinâmica com base nas métricas disponíveis nos fluxos da rede se mostrou uma alternativa viável. O desempenho obtido ante as restrições de processamento, demonstrou que a aplicação de desenvolvida se mostra mais eficaz que as metodologias tradicionais. Já o contorno das limitações de processamento, demonstra que a ferramenta se torna mais eficaz ao identificar os pacotes a serem enviados a um coletor. A alteração de cenários, permitiu ainda verificar que o funcionamento não se altera com a duração dos eventos, demonstrando a constância no seu funcionamento. Da mesma forma, as variações analisadas demostram que não ocorre uma degradação de desempenho ao se escalar tanto os cenários quando o tamanho da janela.

Com isso, a abordagem possibilitou em todas as janelas testadas, validar uma

forma dinâmica e escalável de realizar a orquestração dos dados a serem encaminhados para um coletor INT, somando-se ainda o fato proporcionar uma diminuição do número de dados de telemetria não essenciais enviados aos coletores INT comparação com outras abordagens, sobrecarga em termos de latência do pacote. Assim, entende-se que os objetivos alcançados vão ao encontro dos objetivos inicialmente propostos.

6.2 Trabalhos futuros

Embora este trabalho apresente avanços no desenvolvimento de aplicações INT, acredita-se que muito ainda tem a ser feito no que tange esta emergente tecnologia. Desta forma, destaca-se a seguir algumas direções a serem seguidas na busca por avanços tecnológicos nesta área.

- realizar a definição de casos de uso complementares, buscando ampliar os usos validados da aplicação.
- realizar o registro de software e a posterior divulgação para comunidade, possibilitando a reprodutibilidade dos experimentos conduzidos.

6.3 Publicações

O presente trabalho resultou até o momento em dois artigos científicos publicados, elencados a seguir:

- Ronaldo Canofre M. dos Santos, Arthur F. Lorenzon, Fabio D. Rossi, Marcelo C. Luizelli. "Impacto da programabilidade no plano de dados em SmartNIC". Anais da XIX Escola Regional de Redes de Computadores. ERRC, 2021. (prêmio de melhor artigo na trilha de Pós-graduação)
- Ronaldo Canofre, Ariel G. Castro, Arthur F. Lorenzon, Fábio D. Rossi, Marcelo C. Luizelli. "*Towards Efficient Selective In-Band Network Telemetry Report using SmartNICs*". International Conference on Advanced Information Networking and Applications. AINA, 2022.

REFERÊNCIAS

ADRICHEM, N. L. M. van; DOERR, C.; KUIPERS, F. A. Opennetmon: Network monitoring in openflow software-defined networks. In: **2014 IEEE Network Operations and Management Symposium (NOMS)**. [S.l.: s.n.], 2014. p. 1–8. Citado na página 36.

BELENTANI, L. C.; MARCELLO, J.; FLORIAN, F. A utilização de ferramentas de monitoramento para o otimização do gerenciamento da rede. **Revista Interface Tecnológica**, v. 15, n. 2, p. 99–110, 12 2018. Disponível em: <https://revista.fatectq.edu.br/index.php/interfacetecnologica/article/view/509>. Citado na página 35.

BOSSHART, P. et al. P4: Programming protocol-independent packet processors. Association for Computing Machinery, New York, NY, USA, v. 44, n. 3, p. 87–95, 07 2014. ISSN 0146-4833. Disponível em: <https://doi.org/10.1145/2656877.2656890>. Citado 3 vezes nas páginas 22, 29 e 30.

CAMERA, P. E. **Int flow : aplicação de telemetria em redes definidas por software**. Dissertação (Mestrado) — Programa de Pós-Graduação em Ciência Aplicada, Universidade de Passo Fundo, Passo Fundo,RS, 2020. Disponível em: <http://tede.upf.br:8080/jspui/handle/tede/1906>. Citado 8 vezes nas páginas 27, 29, 32, 34, 36, 37, 43 e 48.

CASE, J. et al. **Simple Network Management Protocol (SNMP)**. Internet Research Task Force (IRTF), 1990. Disponível em: <http://www.rfc-editor.org/rfc/rfc1157.txt>. Citado na página 35.

CASTANHEIRA, L.; PARIZOTTO, R.; FILHO, A. E. S. Flowstalker: Comprehensive traffic flow monitoring on the data plane using P4. In: **2019 IEEE International Conference on Communications, ICC 2019**. Shanghai, China, May 20-24, 2019: IEEE, 2019. p. 1–6. Disponível em: <https://doi.org/10.1109/ICC.2019.8761197>. Citado 2 vezes nas páginas 44 e 48.

CHOWDHURY, S. R. et al. Payless: A low cost network monitoring framework for software defined networks. In: **2014 IEEE Network Operations and Management Symposium (NOMS)**. [S.l.: s.n.], 2014. p. 1–9. Citado na página 36.

CHOWDHURY, S. R.; BOUTABA, R.; FRANÇOIS, J. Lint: Accuracy-adaptive and lightweight in-band network telemetry. In: **IFIP/IEEE International Symposium on Integrated Network Management (IM)**. [S.l.: s.n.], 2021. p. 349–357. Citado 3 vezes nas páginas 36, 44 e 48.

COMPANI, Z. **Zabbix**. 2022. Disponível em: <https://www.zabbix.org/>. Citado na página 35.

CONSORTIUN, P. L. **P4-16 Language Specification**. 2021. Disponível em: <https://p4lang.github.io/p4-spec/docs/P4-16-v1.2.2.html>. Citado na página 30.

CONSORTIUN, P. L. **P4C**. 2022. Disponível em: <https://github.com/p4lang/p4c>. Citado na página 32.

COSTA, L. C. **Balanceamento de carga utilizando planos de dados OpenFlow comerciais**. Dissertação (Mestrado) — Programa de Pós-Graduação em Ciência da

Computação, Universidade Federal de Juiz de Fora, Juiz de Fora,MG, 2016. Disponível em: <https://repositorio.ufjf.br/jspui/bitstream/ufjf/2258/1/leonardochinelatecosta.pdf>. Citado na página 36.

CUGINI, F. et al. P4 in-band telemetry (int) for latency-aware vnf in metro networks. In: **Optical Fiber Communication Conference (OFC) 2019**. Optical Society of America, 2019. p. M3Z.6. Disponível em: <http://www.osapublishing.org/abstract.cfm?URI=OFC-2019-M3Z.6>. Citado 3 vezes nas páginas 36, 45 e 48.

DIMOLIANIS, M.; PAVLIDIS, A.; MAGLARIS, V. A multi-feature ddos detection schema on p4 network hardware. In: **2020 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)**. [S.l.: s.n.], 2020. p. 1–6. Citado 3 vezes nas páginas 36, 46 e 48.

DING, D.; SAVI, M.; SIRACUSA, D. Estimating logarithmic and exponential functions to track network traffic entropy in p4. In: **NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium**. IEEE Press, 2020. p. 1–9. Disponível em: <https://doi.org/10.1109/NOMS47738.2020.9110257>. Citado 2 vezes nas páginas 45 e 48.

ELLER, A. C. E. **Encaminhamento por hardware em redes definidas por software: avaliação experimental utilizanto NetFPGA**. Dissertação (Mestrado) — Programa de Pós-Graduação em Engenharia Elétrica, Universidade do Espeírito Santo, Vitória,ES, 2016. Citado na página 41.

EMMERICH, P. et al. Moongen: A scriptable high-speed packet generator. 10 2014. Citado na página 64.

ENTERPRISES, N. **Nagios**. 2022. Disponível em: <https://www.nagios.org/>. Citado na página 35.

FEAMSTER, N.; REXFORD, J.; ZEGURA, E. The road to sdn: An intellectual history of programmable networks. Association for Computing Machinery, New York, NY, USA, v. 44, n. 2, p. 87–98, 04 2014. ISSN 0146-4833. Disponível em: <https://doi.org/10.1145/2602204.2602219>. Citado 2 vezes nas páginas 25 e 27.

FEFERMAN, D. L. et al. Uma nova revolucao em redes: Programacao do plano de dados com p4. In: **Escola Regional de Informátiva do Piauí (ERIPI), Teresina, Brazil**. [s.n.], 2018. Disponível em: <https://intrig.dca.fee.unicamp.br/wp-content/papercite-data/pdf/feferman2018nova.pdf>. Citado na página 25.

FENG, Y. et al. A new framework for network flow queuing delay prediction based on stream computing. In: **2019 IEEE 5th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)**. Los Alamitos, CA, USA: IEEE Computer Society, 2019. p. 212–217. Citado 3 vezes nas páginas 36, 43 e 48.

FENG, Y. et al. A smartnic-accelerated monitoring platform for in-band network telemetry. In: **2020 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN**. [S.l.: s.n.], 2020. p. 1–6. Citado 3 vezes nas páginas 36, 43 e 48.

GOULART, P. et al. Netfpga: Processamento de pacotes em hardware. **Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC**, v. 2015, p. 1–3, 2015. Citado na página 41.

GROUP, I. T. C. **Cacti**. 2021. Disponível em: <https://www.cacti.net/>. Citado na página 35.

GROUP, P. A. W. **In-band Network Telemetry (INT) Dataplane Specification**. June, 2020. Disponível em: <https://github.com/p4lang/p4-applications/blob/master/docs/INT_v2_1.pdf.> Citado 2 vezes nas páginas 37 e 38.

HALEPLIDIS, E. et al. **Software-Defined Networking (SDN) Layers and Architecture Terminology**. Internet Research Task Force (IRTF), 2015. Disponível em: <https://www.rfc-editor.org/rfc/pdfrfc/rfc7426.txt.pdf>. Citado 2 vezes nas páginas 26 e 27.

HOHEMBERGER, R. et al. Orchestrating in-band data plane telemetry with machine learning. **IEEE Communications Letters**, v. 23, n. 12, p. 2247–2251, 2019. Citado 2 vezes nas páginas 22 e 23.

JEYAKUMAR, V. et al. Millions of little minions: Using packets for low latency network programming and visibility. **ACM SIGCOMM CCR**, ACM New York, NY, USA, v. 44, n. 4, p. 3–14, 2014. Citado na página 21.

JOSHI, R. et al. Burstradar: Practical real-time microburst monitoring for datacenter networks. In: **Proceedings of the 9th Asia-Pacific Workshop on Systems**. New York, NY, USA: Association for Computing Machinery, 2018. (APSys '18). ISBN 9781450360067. Citado na página 21.

KIM, Y.; SUH, D.; PACK, S. Selective in-band network telemetry for overhead reduction. In: **IEEE 7th International Conference on Cloud Networking (CloudNet)**. [S.l.: s.n.], 2018. p. 1–3. Citado 3 vezes nas páginas 36, 44 e 48.

LAPOLLI, A. C.; MARQUES, J. A.; GASPARY, L. P. Offloading real-time ddos attack detection to programmable data planes. In: **2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)**. [S.l.: s.n.], 2019. p. 19–27. Citado 3 vezes nas páginas 36, 46 e 48.

LAUFER, R. P. et al. Negação de serviço: Ataques e contramedidas. **Livro Texto dos Mini-cursos do V Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais**, 2005. Citado na página 64.

LEE, J.-H.; SINGH, K. Switchtree: In-network computing and traffic analyses with random forests. **Neural Computing and Applications**, 11 2020. Citado 3 vezes nas páginas 45, 46 e 48.

LIN, W.-H. et al. Network telemetry by observing and recording on programmable data plane. In: **IFIP Networking Conference (IFIP Networking)**. [S.l.: s.n.], 2021. p. 1–6. Citado 3 vezes nas páginas 36, 44 e 48.

LIU, Z. et al. Netvision: Towards network telemetry as a service. In: **2018 IEEE 26th International Conference on Network Protocols (ICNP)**. [S.l.: s.n.], 2018. p. 247–248. ISSN 1092-1648. Citado na página 21.

LU, B. et al. Ifit: Intelligent flow information telemetry. In: **Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos**. New York, NY, USA: Association for Computing Machinery, 2019. (SIGCOMM Posters and Demos '19), p. 15–17. ISBN 9781450368865. Disponível em: <https://doi.org/10.1145/3342280.3342292>. Citado 2 vezes nas páginas 45 e 48.

MAI, T. et al. In-network intelligence control: Toward a self-driving networking architecture. **IEEE Network**, v. 35, n. 2, p. 53–59, 2021. Citado 2 vezes nas páginas 44 e 48.

MARCUZZO, L. d. C. et al. **Uma arquitetura para o offload parcial de funções virtualizadas de rede em plano de dados programável**. Dissertação (Mestrado) — Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de Santa Marial, Santa Maria, RS, 2020. Disponível em: <http://tede.upf.br:8080/jspui/handle/tede/1906>. Citado na página 33.

MCKEOWN, N. **P4 Runtime – Putting the Control Plane in Charge of the Forwarding Plane**. Open Networking Foundation, 2017. Disponível em: <https://opennetworking.org/news-and-events/blog/p4-runtime-putting-the-control-plane-in-charge-of-the-forwarding-plane/>. Citado na página 33.

MCKEOWN, N. et al. Openflow: Enabling innovation in campus networks. **SIGCOMM Comput. Commun. Rev.**, Association for Computing Machinery, New York, NY, USA, v. 38, n. 2, p. 69–74, 03 2008. ISSN 0146-4833. Disponível em: <https://doi.org/10.1145/1355734.1355746>. Citado 2 vezes nas páginas 26 e 28.

MEGYESI, P. et al. Challenges and solution for measuring available bandwidth in software defined networks. **Computer Communications**, v. 99, p. 48–61, 2017. ISSN 0140-3664. Disponível em: <https://www.sciencedirect.com/science/article/pii/S014036641630648X>. Citado na página 36.

MELO, J. C. F. d. **Gerenciamento adaptativo de redes baseado em contexto e SDN**. Dissertação (Mestrado) — Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de Pernambuco, REcife,PE, 2020. Disponível em: <https://repositorio.ufpe.br/handle/123456789/37905>. Citado na página 35.

MIANO, S. et al. Introducing smartnics in server-based data plane processing: The ddos mitigation use case. **IEEE Access**, v. 7, p. 107161–107170, 2019. Disponível em: <https://ieeexplore.ieee.org/ielx7/6287639/8600701/08789414.pdf>. Citado na página 39.

MORTON, A. C. **Active and Passive Metrics and Methods (with Hybrid Types In-Between)**. Internet Research Task Force (IRTF), 2016. Disponível em: <https://datatracker.ietf.org/doc/html/rfc7799#section-1>. Citado na página 34.

NASCIMENTO, E. B. Uma arquitetura de rede programável para redes orientadas à informação com replicação de conteúdo em nuvens privadas. São Cristóvão, SE, 2018. Citado na página 27.

NETFPGA, G. O. **NetFPGA Guide**. 04, 2013. Disponível em: <https://github.com/NetFPGA/netfpga/wiki/Guide>. Citado na página 41.

NETFPGA, G. O. **P4 NetFPGA**. 11, 2018. Disponível em: <https://github.com/NetFPGA/P4-NetFPGA-public/wiki>. Citado na página 42.

NETRONOME, S. I. **NFP-4000 Theory of Operation**. 2016. Disponível em: <https://www.netronome.com/static/app/img/products/silicon-solutions/WP_NFP4000_TOO.pdf>. Citado 2 vezes nas páginas 39 e 40.

NETRONOME, S. I. **Programming NFP with P4 and C**. 2017. Disponível em: <https://www.netronome.com/media/redactor_files/WP_Programming_with_P4_and_C.pdf>. Citado na página 40.

PAN, T. et al. Int-path: Towards optimal path planning for in-band network-wide telemetry. In: **IEEE INFOCOM 2019 - IEEE Conference on Computer Communications**. [S.l.: s.n.], 2019. p. 487–495. Citado 3 vezes nas páginas 21, 22 e 23.

POSTEL, J. **Internet Control Message Protocol**. RFC Editor, 1981. (Request for Comments, 792). Disponível em: <https://rfc-editor.org/rfc/rfc792.txt>. Citado na página 35.

QUEIROZ, W.; CAPRETZ, M. A.; DANTAS, M. An approach for sdn traffic monitoring based on big data techniques. **Journal of Network and Computer Applications**, v. 131, p. 28–39, 2019. ISSN 1084-8045. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1084804519300244>. Citado na página 36.

REZENDE, P. H. A. **Extensões na arquitetura SDN para o provisionamento de QoS através do monitoramento e uso de múltiplos caminhos**. Dissertação (Mestrado) — Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de Ubelândia, Uberlândia,MG, 2016. Disponível em: <https://repositorio.ufu.br/handle/123456789/17550>. Citado na página 36.

SANTOS, R. C. M. d.; LUIZELLI, M. C. Impacto da programabilidade no plano de dados em smartnic. In: **Escola Regional de Redes de Computadores (ERRC), RS, Brazil**. [s.n.], 2021. Disponível em: <https://temQueCorrigirEssaUrlQuandoPublicarOsAnais.com.br/arquivo.pdf>. Citado na página 61.

SAQUETTI, M. et al. Toward in-network intelligence: Running distributed artificial neural networks in the data plane. **IEEE Communications Letters**, v. 25, n. 11, p. 3551–3555, 2021. Citado 2 vezes nas páginas 22 e 23.

SILVA, E. F. d. et al. **SDNMonitor: um serviço de monitoramento de tráfego em redes definidas por software**. Dissertação (Mestrado) — Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de Sergipe, São Cristóvão,SE, 2016. Disponível em: <http://tede.upf.br:8080/jspui/handle/tede/1906>. Citado na página 34.

SINGH, S. K. et al. Revisiting heavy-hitters: Don't count packets, compute flow inter-packet metrics in the data plane. In: **ACM SIGCOMM Poster**. New York, NY, USA: ACM, 2020. p. 1–4. Citado na página 21.

SONG, H. Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane. In: . New York, NY, USA: Association for Computing Machinery, 2013. p. 127–132. ISBN 9781450321785. Disponível em: <https://doi.org/10.1145/2491185.2491190>. Citado na página 29.

SUH, D. et al. Flexible sampling-based in-band network telemetry in programmable data plane. **ICT Express**, v. 6, n. 1, p. 62–65, 2020. ISSN 2405-9595. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2405959519302358>. Citado 2 vezes nas páginas 44 e 48.

SUH, J. et al. Opensample: A low-latency, sampling-based measurement platform for commodity sdn. In: **2014 IEEE 34th International Conference on Distributed Computing Systems**. [S.l.: s.n.], 2014. p. 228–237. Citado na página 36.

TAN, L. et al. In-band network telemetry: A survey. **Computer Networks**, v. 186, p. 29, 08 2021. Citado 3 vezes nas páginas 34, 36 e 37.

VESTIN, J. et al. Programmable event detection for in-band network telemetry. In: **IEEE 8th International Conference on Cloud Networking (CloudNet)**. [S.l.: s.n.], 2019. p. 1–6. Citado 2 vezes nas páginas 45 e 48.

VESTIN, J. et al. Toward in-network event detection and filtering for publish/subscribe communication using programmable data planes. **IEEE Transactions on Network and Service Management**, v. 18, n. 1, p. 415–428, 2021. Citado 2 vezes nas páginas 61 e 62.

VIEGAS, P. et al. The actual cost of programmable smartnics: diving into the existing limits. In: BAROLLI, L. et al. (Ed.). **Advanced Information Networking and Applications**. [S.l.]: Springer International Publishing, 2021. p. 381–392. Citado 2 vezes nas páginas 23 e 24.

VIEGAS, P. B. et al. The actual cost of programmable smartnics: Diving into the existing limits. In: BAROLLI, L.; WOUNGANG, I.; ENOKIDO, T. (Ed.). **Advanced Information Networking and Applications**. Cham: Springer International Publishing, 2021. p. 181–194. ISBN 978-3-030-75100-5. Citado na página 59.

WANG, S. et al. Flowtrace: measuring round-trip time and tracing path in software-defined networking with low communication overhead. **Frontiers of Information Technology & Electronic Engineering**, v. 18, p. 206–219, 02 2017. Citado na página 36.

WRAY, S. **The Joy Of Micro-C**. Open-NFP, 2014. Disponível em: <https://cdn.open-nfp.org/media/documents/the-joy-of-micro-c_fcjSfra.pdf>. Citado 2 vezes nas páginas 40 e 62.

XAVIER, B. M. et al. Programmable switches for in-networking classification. In: **IEEE INFOCOM 2021 - IEEE Conference on Computer Communications**. [S.l.: s.n.], 2021. p. 1–10. Citado 2 vezes nas páginas 46 e 48.

XIONG, Z.; ZILBERMAN, N. Do switches dream of machine learning? toward in-network classification. In: . New York, NY, USA: Association for Computing Machinery, 2019. (HotNets '19), p. 25–33. ISBN 9781450370202. Disponível em: <https://doi.org/10.1145/3365609.3365864>. Citado 3 vezes nas páginas 45, 46 e 48.

YU, C. et al. Software-defined latency monitoring in data center networks. In: MIRKOVIC, J.; LIU, Y. (Ed.). **Passive and Active Measurement**. Cham: Springer International Publishing, 2015. p. 360–372. ISBN 978-3-319-15509-8. Citado na página 36.

YU, M. Network telemetry: Towards a top-down approach. **SIGCOMM Comput. Commun. Rev.**, Association for Computing Machinery, New York, NY, USA, v. 49, n. 1, p. 11–17, 02 2019. ISSN 0146-483. Disponível em: <https://doi.org/10.1145/3314212.3314215>. Citado na página 37.

ZANETTI, A. B. **Visual-INT : coleta, processamento e visualização granular de metadados INT para gerência SDN**. Dissertação (Mestrado) — Programa de Pós-Graduação em Computação Aplicada, Instituto de Ciências Exatas e Geociências – ICEG, Passo Fundo, RS, 2020. Disponível em: <http://tede.upf.br:8080/jspui/handle/tede/1904>. Citado na página 25.

ZHANG, X. et al. A two-way link loss measurement approach for software-defined networks. In: . [S.l.: s.n.], 2017. p. 1–10. Citado na página 36.

ZHAO, Y.; ZHANG, P.; JIN, Y. Netography: Troubleshoot your network with packet behavior in sdn. In: **NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium**. [S.l.: s.n.], 2016. p. 878–882. Citado na página 36.

ZHOU, Y. et al. Flow event telemetry on programmable data plane. In: **Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication**. New York, NY, USA: Association for Computing Machinery, 2020. (SIGCOMM '20), p. 76–89. ISBN 9781450379557. Disponível em: <https://doi.org/10.1145/3387514.3406214>. Citado 2 vezes nas páginas 43 e 48.

ZILBERMAN, N. et al. Netfpga sume: Toward 100 gbps as research commodity. **Micro, IEEE**, v. 34, p. 32–41, 09 2014. Citado na página 42.

Apêndices

APÊNDICE A – IMPACTO DA PROGRAMABILIDADE NO PLANO DE DADOS EM SMARTNIC

Impacto da programabilidade no plano de dados em SmartNIC

Ronaldo Canofre M. dos Santos, Arthur F. Lorenzon, Fabio D. Rossi, Marcelo C. Luizelli

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

Instituto Federal Farroupilha (IFFAR) Alegrete – RS – Brasil

{canofre, arthurlorenzon, marceloluizelli}@unipampa.edu.br

fabio.rossi@iffarroupilha.edu.br

Resumo. *A programabilidade no plano de dados é um novo paradigma que através da linguagem P4, proporciona uma utilização mais eficiente dos dispositivos, sendo as SmartNICs disponíveis atualmente, um fator importante para sua adoção, devido a possibilidade de utilização em um ambiente real. No entanto, existem custos e limitações atreladas a este conjunto que requerem uma certa atenção. Neste trabalho é realizada avaliação utilizando um conjunto específico de configurações a fim de quantificar as limitações de desempenho existentes ainda não avaliadas. Os resultados mostram que a taxa de encaminhamento de pacotes pode alcançar uma degradação superior a 50%.*

1. Introdução

A programabilidade de rede surgiu como parte da busca de novos paradigmas para gerência de redes [Nascimento 2018], se tornando uma alternativa a rigidez empregada atualmente na atualização dos dispositivos de rede, buscando mais flexibilidade, escalabilidade e controle, obtendo melhor aproveitamento atrelado a um menor custo [Feferman et al. 2018]. A sua utilização já vem sendo abordada em trabalhos realizados nas últimas duas ou três décadas, tais como redes ativas, redes ATM programáveis e em propostas para controle e separação do plano de dados, como NCP e RCP [Kreutz et al. 2015] e mais recentemente a arquitetura SDN (*Software Defined Networking*).

A ideia de tornar o plano de dados programável foi apresentada em 2014 por [Bosshart et al. 2014], como uma alternativa às limitações apresentadas pelos controladores das redes SDN, viabilizando uma opção mais simples, independente de protocolo e limitação de cabeçalhos pré-definidos, através da utilização de uma linguagem de alto nível denominada P4 (*Programming Protocol-independent Packet Processors*). Desde então, trabalhos baseados na programabilidade dos planos de dados vem sendo desenvolvidos, abordando inúmeros e distintos objetivos, no entanto, pouco se aborda sobre o impacto dessa programabilidade na capacidade de encaminhamento ou *forwarding rate*.

Uma das possíveis causas para esta situação pode ser relacionada ao desenvolvimento e realização de testes em ambientes virtuais, possibilitando assim a ampliação de recursos, o que não se torna possível quando aplicada sobre dispositivos programáveis como interfaces SmartNIC ou NetFPGA. Assim, este trabalho busca realizar uma análise do impacto do uso de programas P4 sobre interfaces Agilio CX da Netronome,

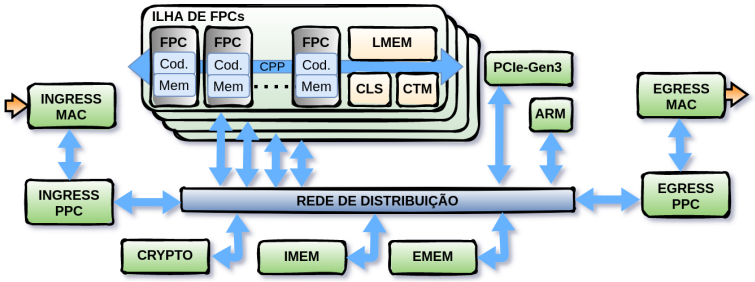


Figura 1. Arquitetura NFP4000

desenvolvidas com a arquitetura NFP4000 a fim de demonstrar os efeitos do aumento de processamento na programação voltada para o plano de dados sobre o *forwarding rate*.

O restante deste trabalho está organizado da seguinte forma. Apresenta-se na Seção 2 a arquitetura da interface de rede utilizada, enquanto que na Seção 3, discute-se os trabalhos relacionados. Na Seção 4, apresenta-se a implementação, a metodologia empregada e o ambiente de avaliação. Os resultados obtidos são apresentados na Seção 5 e, por fim, a Seção 6 apresenta as considerações finais e perspectivas futuras.

2. SmartNICs Netronome

As SmartNICs ou NICs inteligentes, podem se consideradas interfaces completamente programáveis, com um processamento paralelo e multi core baseado em uma abordagem SoC (*Systen-on-chip*), a qual combina uma ou mais CPUs com as funções padrões da NIC [Miano et al. 2019]. A SmartNIC NFP4000 [Netronome Systems 2016] utilizada, apresenta uma arquitetura modular composta por ilhas, tais como de memórias e de unidades de processamento, vide figura 1. Estas unidades são máquinas de 32 *bits* denominadas FPCs (*Flow Processing Cores*), com espaços distintos para armazenamento de dados e instruções, permitindo que cada unidade execute um conjunto diferenciado de ações.

As FPCs contam também com 8 *threads* que atuam sobre o mesmo código e memória local, trocando informações entre os componentes através de um barramento denominado CPP (*command-push-pull*). Cada FPC é composta ainda por uma memória local com 1024 palavas de 32 *bits* e três conjuntos de registradores, sendo 256 de propósito geral, divididos entre as 8 *threads*, 512 de transferência, utilizados para trafegar dados no barramento CPP e 128 do tipo *next-neighbor*, utilizados na comunicação com FPCs vizinhas na mesma ilha. A interface contém ainda 5 ilhas de processamento compostas por 12 FPCs e uma memória local (LMEM) de 4KB, podendo possuir ainda outros dois tipos de memória: CLS de 64KB, para dados necessários à maioria dos pacotes e CTM de 256 KB, para cabeçalhos de pacotes e coordenação entre FPCs e outros sub-sistemas.

Externamente as ilhas de processamento, cada interface possui ainda mais duas memórias: IMEM de 4MB, destinada ao corpo dos pacotes e compartilhamento de tabelas médias (2 por interface) e (EMEM) de 3MB + DRAM, para compartilhamento de grandes tabelas (3 por *chip*) [Netronome Systems 2020]. A programação destas interfaces pode ser realizada através de linguagens de alto nível como P4 e Micro-C através de um compilador proprietário, apresentando no entanto algumas restrições e diferenças comportamentais com relação ao P4 modelo. A linguagem Micro-C consiste em um subconjunto do Standard C, principalmente devido a limitações relacionadas aos tipos de dados suportados, funções e cabeçalhos de biblioteca padrão.

3. Trabalhos relacionados

Esforços consideráveis voltados ao uso eficiente de aplicativos em SmartNICS tem sido realizados, tais como DAIET [Sapio et al. 2017] e Clara [Qiu et al. 2020], que assim como a grande maioria dos trabalhos, se destinam a avaliação do descarregamento da aplicação no plano de dados, não levando em consideração as limitações atuais das SmartNICs, deixando de lado a avaliação do impacto causado pelas estruturas e pelo processamento realizado. Por outro lado, alguns trabalhos tem focado na análise das estruturas e componentes de programas P4 e no impacto sobre o desempenho do processamento em interfaces inteligentes.

Em [Harkous et al. 2019], é apresentada uma avaliação do impacto na latência de processamento de pacotes realizada em diferentes programas P4, realizando o aumento gradativo da complexidade através da inclusão de alguns blocos como analisador e controle, buscando identificar as variáveis mais influentes para prever a latência do pacote. Já em [Viegas et al. 2021], é realizada uma análise do *throughpt* e latência, através da elevação quantidade das métricas utilizadas, analisando o volume de operações em registradores, de acessos em tabelas de fluxo, de recirculação de pacotes, de funções de criptografia e de operações aritméticas.

No entanto, tais trabalhos não ponderam as questões referentes a adição de campos mais simples, como cabeçalhos adicionais ou personalizados, não sendo também realizadas avaliações relativas a utilização de componentes externos, como o uso de módulos em Micro-C ou análises referentes ao controle de concorrência e o impacto trazido com seu uso. Ademais, tais trabalhos não consideram a limitação de utilização de processadores, até o momento necessária para obtenção de uma corretude na computação de valores em todos os pacotes, os quais implicam na diminuição do *throughpt*. Assim sendo, este trabalho busca contribuir com as avaliações do impacto do P4 em SmartNICs através da análise dos pontos supracitados.

4. Implementação e metodologia

Como ponto de partida, utilizou-se o código de encaminhamento de pacotes disponibilizado pelo P4.org (caso 1[1]), a partir do qual foram realizadas inclusões de estruturas e funcionalidades de forma complementar ao caso anterior, sendo analisando o impacto sobre a quantidade de pacotes encaminhados a cada nova alteração. Para o caso 2, foi adicionado um cabeçalho UDP juntamente com sua inicialização, seguido da adição de um cabeçalho personalizado denominado APF compostos de 4 campos de 32 *bits* e um campo de 2 *bits* (caso 3) e por fim de um cabeçalho de metadados contento dois campos de 64 *bits* com registros de *timestamp* de ingresso no dispositivo e atual (caso 4), todos sem utilização no processamento do plano de dados.

Posteriormente foi atribuído ao cabeçalho APF a transposição de informações entre os estágios do processamento (caso 5), seguido da recuperação do tamanho do pacote, cálculo da latência e seu respectivo armazenamento em um registrador de 32 *bits* (caso 6). Devido as limitações de processamentos apresentadas pela interface da Netronome ao se realizar uma correta computação de latência para o grande volume de dados enviados, foi adicionada a implementação de um módulo em Micro-C, responsável pelo cálculo da média entre as métricas disponíveis (caso 7).

[1] https://github.com/p4lang/tutorials/tree/master/exercises/basic

A partir desta programação, buscando também avaliar o impacto das metodologias disponíveis para o tratamento do paralelismo, foi adicionado um contador de pacotes e realizada uma avaliação com o uso do mecanismos de controle de concorrência mutex (caso 8) e semáforo (caso 11), sendo o primeiro disponibilizado pelas bibliotecas da Netronome e segundo programado em Micro-C. Considerando que nestas avaliações é realizado o bloqueio da execução para todos os pacotes, foram também analisadas reduções na sua utilização, realizando a aplicação dos bloqueios a cada 5 e 100 pacotes, tanto para mutex (casos 9 e 10) como para semáforo (casos 12 e 13).

O ambiente de desenvolvimento consiste em dois servidores com processador AMD Ryzen 7 3800X com 8 cores e 24GB de RAM, com interfaces Netronome SmartNIC Agilio CX 10 Gbit/s, sendo um deles utilizado para carregamento dos programas P4, atuando como comutador, e o outro utilizado como gerador de tráfego. Foi utilizado o gerador de pacotes da Netronome, através do gerador de trafego DPDK[2] disponibilizado pelo MoonGen [Emmerich et al. 2014]. Foram enviados pacotes IPv4 de 64B com endereços de origem e destino fixos por intervalos de 10 segundos, sendo cada resultado obtido através de uma média simples de três execuções. Foram também aplicadas reduções para limitação da quantidade de processadores utilizados, reduzindo a capacidade de encaminhamento da interface, a qual gira em torno de 10 Gbps para cerca de 2.5 Gbps.

5. Resultados

A figura 2 apresenta os resultados dos casos 1 à 8, sendo possível perceber a existência do impacto na taxa de encaminhamento em cada caso, mesmo que mínimo. No caso 2 à 4 é possível perceber uma leve degradação a cada inclusão de um novo cabeçalho, resultando em um o impacto de 6.34% somente com a adição de três novos cabeçalho, demonstrando a existência de uma degradação relacionada a quantidade destes campos. Nos casos 5 e 6 são apresentadas as maiores variações observadas, oriundas da inclusão de operações no plano de dados. No caso 5, o custo da utilização do cabeçalho APF é de 15.17% em relação ao caso anterior, elevando-se para 20.55% quando comparado com o processamento inicial.

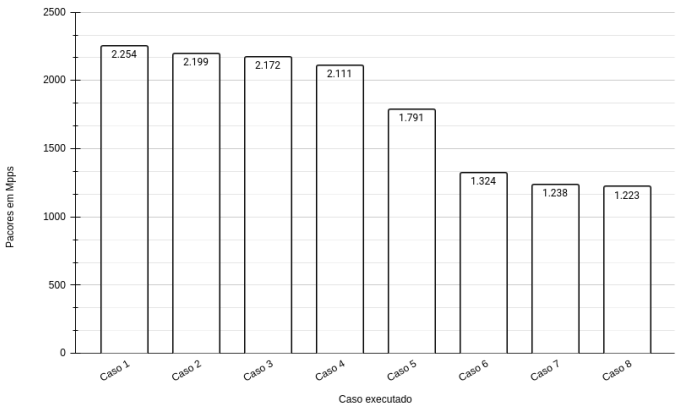


Figura 2. Quantidade de pacotes por caso executado.

No caso 6, o impacto apresentado em realação a execução anterior é mais significativo, cerca de 26%, elevando-se para cerca de 41% em relação ao processamento

[2]https://www.dpdk.org/

inicial, praticamente o dobro do impacto anterior (20.55%), demonstrando a existência de degradação atrelada ao processamento empregado, visto que no caso 5 somente ocorre a inicialização de campos no cabeçalho APF e no caso 6 são realizados além da inicialização de campos com cálculos, o armazenamento dos valores em registradores internos do plano de dados. Já nos casos seguintes, onde ocorre a inclusão do módulo em Micro-C (caso 7) e a utilização de um controle de concorrência com mutex (caso 8), é possível verificar que o impacto com relação aos casos anteriores não é extremamente elevado, gerando no entanto um custo acumulado de mais de 45% na quantidade de pacotes encaminhando ao final de todos os incrementos de processamento realizados até o caso 8.

A tabela 1 apresenta os resultados dos testes de controle de concorrência implementados com mutex e semáforo, sendo possível avaliar que são muito próximos os impactos causados por ambos os casos (8 e 11) sobre o processamento, onde o uso do semáforo apresenta um custo de apenas 0.6% a mais que o uso do mutex, em comparação ao processamento inicial. No entanto, ao se implementar uma redução no uso dos bloqueios, é possível avaliar que em ambos os métodos utilizados ocorre um acréscimo no custo de processamento, resultando em uma degradação superior a 50% em comparação com o processamento inicial, ao contrário do esperado. Por fim, a aplicação do bloqueio de concorrência em um intervalo maior de pacotes (casos 10 e 13) não resulta em uma diferença significativa da quantidade de pacotes encaminhados, mantendo resultados idênticos ou muito próximos.

Tabela 1. Encaminhamento de pacotes por bloqueio implementado.

Caso	Taxa de encaminhamento	Descrição
Caso 8	1.223 Mpps	mutex
Caso 9	1.101 Mpps	mutex a cada 5 pacotes
Caso 10	1.102 Mpps	mutex a cada 100 pacotes
Caso 11	1.210 Mpps	semáforo
Caso 12	1.005 Mpps	semáforo a cada 5 pacotes
Caso 13	1.005 Mpps	semáforo a cada 100 pacotes

6. Conclusão

Neste trabalho analisou-se o impacto da adição de componentes em um programa P4 sobre a taxa de encaminhamento de pacotes em interfaces SmarNIC. Através dos testes realizados foi possível observar a ocorrência de degradação em todos os casos, sendo mais evidenciada na inclusão de processamento no plano de dados, ocorrendo em uma escala inferior quando da adição de cabeçalhos e adição de módulos externos. Relacionado as avaliações de controle de concorrência, foi possível avaliar que a utilização de mutex ou semáforo resultam em um impacto muito próximo, sendo mais prejudicial a aplicação de bloqueios parciais em ambos os controles.

Os resultados obtidos servirão também como parâmetro para comparações e análise, no desenvolvimento de um protótipo de monitoramento e detecção preditiva de anomalias sob o mesmo ambiente utilizado. No entanto, algumas questões podem ainda ser avaliadas em trabalhos futuros, tais como a possibilidade dos resultados estarem relacionados como a interface utilizada, sendo a comparação com outras NICs inteligentes uma avaliação a ser realizada. Outro ponto a ser analisado futuramente, denota sobre a existência ou não da estabilidade na degradação.

Agradecimentos

Este trabalho foi parcialmente financiado por: Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) – 2018/23092-1, 2020/05183-0, 2020/05115-4; Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (FAPERGS) – 19/2551-0001266-7, 19/2551-0001224-1, 19/2551-0001689-1, 21/2551-0000688-9; e pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) – 427814/2018-9.

Referências

Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., and Walker, D. (2014). P4: Programming protocol-independent packet processors. 44(3):87–95.

Emmerich, P., Wohlfart, F., Raumer, D., and Carle, G. (2014). Moongen: A scriptable high-speed packet generator.

Feferman, D. L., Mejia, J. S., Saraiva, N. F., and Rothenberg, C. E. (2018). Uma nova revolucao em redes: Programacao do plano de dados com p4. In *Escola Regional de Informátiva do Piauí (ERIPI), Teresina, Brazil*.

Harkous, H., Jarschel, M., He, M., Priest, R., and Kellerer, W. (2019). Towards understanding the performance of p4 programmable hardware. In *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pages 1–6.

Kreutz, D., Ramos, F. M. V., Veríssimo, P. E., Rothenberg, C. E., Azodolmolky, S., and Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76.

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.

Nascimento, E. B. (2018). Uma arquitetura de rede programável para redes orientadas à informação com replicação de conteúdo em nuvens privadas.

Netronome Systems, I. (2016). NFP-4000 Theory of Operation. Disponível em: `https://bit.ly/wpNfp4000`.

Netronome Systems, I. (2020). Netronome NFP-4000 Flow Processor. Disponível em: `https://bit.ly/pbNfp4000`.

Qiu, Y., Kang, Q., Liu, M., and Chen, A. (2020). Clara: Performance clarity for smartnic offloading. HotNets '20, page 16–22, New York, NY, USA. Association for Computing Machinery.

Sapio, A., Abdelaziz, I., Aldilaijan, A., Canini, M., and Kalnis, P. (2017). In-network computation is a dumb idea whose time has come. HotNets-XVI, page 150–156, New York, NY, USA. Association for Computing Machinery.

Viegas, P. B., de Castro, A. G., Lorenzon, A. F., Rossi, F. D., and Luizelli, M. C. (2021). The actual cost of programmable smartnics: Diving into the existing limits. In Barolli, L., Woungang, I., and Enokido, T., editors, *Advanced Information Networking and Applications*, pages 181–194, Cham. Springer International Publishing.

APÊNDICE B – TOWARDS EFFICIENT SELECTIVE IN-BAND NETWORK TELEMETRY REPORT USING SMARTNICS

Towards Efficient Selective In-Band Network Telemetry Report using SmartNICs

Ronaldo Canofre, Ariel G. Castro, Arthur F. Lorenzon, Fábio D. Rossi, Marcelo C. Luizelli

Abstract

In-band Network Telemetry (INT) is a promising network monitoring approach that allows broad and fine-grained network visibility. However, when a massive volume of telemetry data is reported to an INT collector, there might overload the whole network infrastructure, while still degrading the performance of packet processing at the INT sink node. As previously reported in the literature, programmable devices – in particular, SmartNICs – have strict constraints in terms of processing and memory. In this work, we propose to design and implement a lightweight Exponentially Weighted Moving Average based mechanism inside the SmartNIC data plane in order to assist the decision-making process of reporting INT data. By evaluating our solution in state-of-the-art SmartNICs, we show that our proposal can decrease the number of nonessential telemetry data sent to INT collectors by up to 16X compared to the de-facto INT approach while presenting minor overhead in terms of packet latency.

1 Introduction

In-band Network Telemetry is a promising near real-time network monitoring approach [12, 17, 22] that enables wide and fine-grained network visibility. In a nutshell, INT consists of instrumenting the collection of low-level network

Ronaldo Canofre, Ariel G. Castro, Arthur F. Lorenzon, Marcelo C. Luizelli
Federal University of Pampa (UNIPAMPA)
Alegrete, Brazil, e-mail: {canofre,ariel.aluno,arthurlorenzon,marceloluizelli}@unipampa.edu.br

Fabio D. Rossi
Federal Institute Farroupilha (IFFAR)
Alegrete, Brazil, e-mail: fabio.rossi@iffarroupilha.edu.br

monitoring statistics directly from the data plane – allowing network operators/monitoring applications to be fed with an unprecedented level of information. Examples of such in-network statistics include data plane metadata (e.g., per-packet processing time, or queue utilization), and/or custom-made ones (e.g., network flow inter-packet gap [24]). Over the last years, INT has been successfully applied to a series of use cases [18], including the identification of short-lived network behaviors [13] and network anomalies [11].

In the classic hop-by-hop INT specification (i.e., INT-MD (eMbed Data)[1]), an INT source node embeds instructions into production network packets typically using either unused header fields (e.g., IPv4 options) or by re-encapsulating the network traffic (e.g., using INT encapsulation). Then, INT transit nodes embed metadata to these packets according to the instructions given by the INT source. Last, an INT sink node strips the instruction out of the packet and sends the accumulated telemetry data to an INT collector. Figure 1 illustrates the whole INT procedure. In this example, a packet from network flow f_1 is used to collect INT data from forwarding devices A to F. Recently, investigations have made the first efforts to efficiently orchestrate how INT metadata are collected by network packets [22, 2, 11, 5] in order to increase network visibility and timely detect network events. That includes, for instance, selecting the appropriate network flows/packet to collect the right network telemetry metadata in the network infrastructure. This problem has been proved to be NP-hard since packets might have different spare capacities (e.g., limited by the MTU data link) [19].

Despite these efforts, little has yet been done to efficiently and wisely report the collected telemetry data to an INT collector [23]. In the case where all the telemetry data is reported to an INT collector, there might lead to *(i) an excessive usage of network links between the INT sink and the INT collector*. For instance, if we consider a 10 Gbit/s network link sending 64-Bytes packets (i.e., 14.88 Mpps) and collecting 1 Byte per INT node transit along the way, the volume of network traffic needed to be reported per second would be 118 Mbit $*$ hops (path length). In fact, this volume of reported data can increase substantially if we assume the canonical reference architecture for programmable devices[2] where each device has at least 30 Bytes of metadata; *(ii) performance degradation on packet processing capabilities at the INT sink* due to the usage of packet cloning/recirculation primitives inside the data plane. To send the network packet to the INT collector (or part of it), programmable devices have to rely on packet recirculation/cloning primitives to duplicate the packet – which dramatically reduces the performance in terms of throughput and latency [27]; and *(iii) the overwhelm of the INT collector application* with INT data packets.

To fill in this gap, in this work we propose a selective INT report mechanism entirely implemented using a SmartNIC. As previously reported by [27], programmable devices have stringent constraints in terms of processing capa-

[1] INT specification: `https://github.com/p4lang/p4-applications/blob/master/docs/INT_v2_1.pdf`

[2] `https://github.com/p4lang/p4c/blob/main/p4include/v1model.p4`

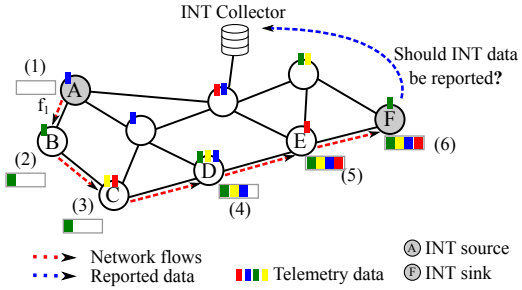


Fig. 1 Overview of the problem.

bilities (e.g., lack of floating-point operations) and memory usage limitations. We assume that the INT sink node is on top of a SmartNIC and, therefore, the decision whether report telemetry data or not is upon the NIC. Our proposed mechanism utilizes a lightweight Exponentially Weighted Moving Average inside the data plane. For that, we implemented it using P4 language and Micro-C routines to allow more complex operations inside the data plane. By performing an extensive performance evaluation using SmartNICs, we show that our proposed approach can reduce the amount of non-important telemetry data sent to INT collector by up to 16X (when compared to the Classical INT), while introducing negligible overhead in terms of packet latency.

The main contributions of this paper can be summarized as:

- an in-network mechanism implemented in state-of-the-art SmartNICs to wisely decide when to report INT metadata;
- a discussion of current limitations on implementing in-network computing in SmartNIC architectures; and
- an open-source code in order to foster reproducibility.

The remainder of this paper is organized as follows. In Section 2, we describe the SmartNIC architecture used in this work. In Section 3, we introduce our proposed approach. In Section 4, we discuss the obtained results. In Section 5, we overview the recent literature regarding in-band network telemetry, and. Last, in Section 6, we conclude this paper with final remarks.

2 Background

Cutting-edge programmable NICs (named SmartNICs) rely their architectures either on (i) multi-threaded, multi-core flow processor units or (ii) on FPGAs (Field Programmable Gate Arrays) to meet the increasing and strict demand. We concentrate our analysis on the general architectural elements of the Netronome SmartNIC architecture [21] – which is used afterward in our performance experiments – and rely on a multi-core architecture.

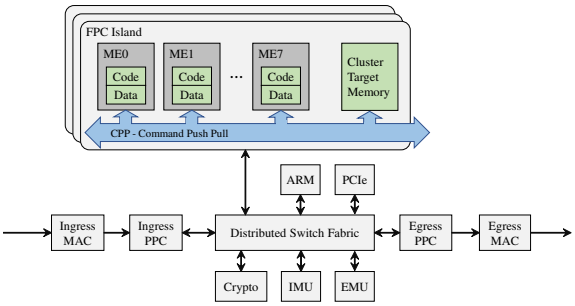


Fig. 2 An overview of the Netronome SmartNIC architecture [27].

The SmartNIC Netronome NFP4000 architecture manages its flow processing cores (FPC) in multiple islands (Figure 2). Each FPC includes eight Micro Engines (MEs) as a particular processor keeping its own instruction store (*code*) and local memory (*data*). Therefore, every ME in the architecture can run code with all other MEs in parallel. To support this feature, each ME holds 8 threads that can be used for cooperative multithreading in the manner that, at any given moment, at most, one thread is executing code from the same program. It means that each FPC handles at most eight parallel threads at 1.2Ghz (one thread per ME). In each FPC, local memory comprises 32-bit registers, shared between all eight threads. Such registers are separated into: (*i*) general-purpose registers (256 32-bits registers) – used by default to store any register of up to 32-bits size; (*ii*) transfer registers (512 32-bits registers) – used for copying register over the interconnection bus (e.g., from or to other FPCs or memories); (*iii*) next-neighbor registers (128 32-bits registers) – used mostly to intercommunicate with adjacent FPCs; and (*iv*) local memory (1024 32-bit registers) – which is a little bit slower than general register. When there is a demand for more memory than available space in local FPC registers, variables are automatically and statically assigned to other in-chip memory hierarchies. Further, there are other sorts of memory available to FPCs: (*i*) Cluster Local Scratch (CLS) (20-50 cycles); (*ii*) Cluster Target Memory (CTM) (50-100 cycles); (*iii*) Internal Memory (IMEM) (120-250 cycles); and (*iv*) External Memory (EMEM) (150-590 cycles). For further details, the interested reader is referred to [21].

As packets are acquired from the network, an FPC thread picks up the packets and processes them. Extra threads are assigned to new packets as they arrive. For example, the SmartNIC NFP-4000 supports up to 60 FPCs, which enables the process of up to 480 packets simultaneously. The SmartNIC allows to program it directly using Micro-C language (i.e., a subset of C language) or using high-level domain-specific languages such as P4 [3]. The code is then compiled and statically assigned to a particular subset of FPC.

3 ETA: Early Network Telemetry Flow Analyzer Approach

In this section, we first define the model used by our proposed approach to decide whether or not network telemetry data is sent to an INT collector. Then, we describe how it is implemented in a SmartNIC and discuss existing limitations and challenges.

3.1 Model and Problem Definition

We consider that a programmable forwarding device $d \in D$ has $N \in \mathbb{N}^+$ available metadata to be collected by an INT-enabled packet p and that such packet has a limited available capacity to carry up to $M \in \mathbb{N}^+$ of such N items ($M \geq N$). For simplicity, we assume that all devices D have the same telemetry information and that an INT-enabled packet p can only collect telemetry data atomically, that is, it either collects all N metadata from $d \in D$ or none of them. Packet p can only collect once the same subset of telemetry data from the device d. We assume that the INT source instructs the packet p correctly according to a given algorithm (e.g., [19]).

Consider that a packet p has collected $M' \subseteq M$ telemetry data along its routing path – which comprises a subset of D devices. When the packet p gets to the INT sink node, the question to be answered is: *should it be sent to the INT collector or not?* To answer this question, the INT sink node computes (i) a weighted average of metadata M' collected by packet p and (ii) an exponential weighted moving average. The former tends to weigh the collected telemetry data differently according to its importance. For instance, the processing time (or the queue utilization) metadata might be more important to be considered in the decision process than the packet size. In turn, the latter tends to keep in memory the observed behavior of the latest received telemetry information over time.

Upon a received packet p in the INT sink node, it extracts the M' collected telemetry metadata and computes a weighted average per packet $A_p = \sum_{i=1}^{M'} w_i \cdot M_i$ (Eq. 1), where $w_i \in [0, 1]$ is the i-*th* weight given to the telemetry data. We further assume that $\sum_{i=1}^{M} w_i = 1$ and that M_i corresponds to the i-*th* collected data. Such individual averages A_p are then summing up into a accumulated weighted average metric within a given time window W (we discuss this design choice next). The time window W is defined for simplicity as a predefined number of packets. However, it can be extended to other metrics such as a time interval.

Then, the accumulated weighted average of a given window W is given by $A_w = \frac{\sum_{p=1}^{W} A_p}{W}$ (Eq. 2). Similarly, the exponential weighted moving average is obtained by $A_e = \alpha \cdot A_w + (1 - \alpha) \cdot A_e$ (Eq. 3) where $\alpha \in [0, 1]$ comprises the importance given by the elements obtained in the last window W versus

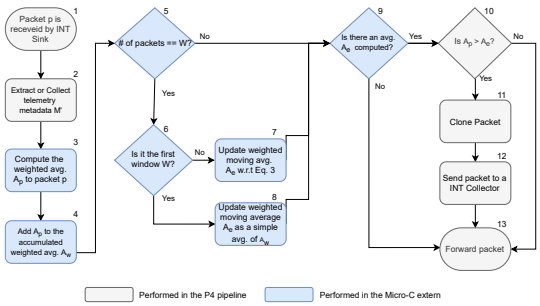


Fig. 3 Overview of the proposed P4+Micro-C pipeline approach.

the historical knowledge maintained by the moving average A_e. Observe that higher values assigned to α prioritize the behavior on the latest window, while lower values prioritize the observed behavior over time. The decision-making process is made in a per-packet manner; however, the decision-making metrics (e.g., A_e) are updated in a time-window manner.

3.2 Design and Implementation in a SmartNIC

Figure 3 illustrates an overview of the proposed approach pipeline implementation. Our approach utilizes the reference V1Model architecture to programmable forwarding devices as the basis to implement/add such functionalities into the Netronome SmartNIC. Also, our approach is based on P4-16 and in Micro-C languages.

Upon a packet is received by the SmartNIC, the packet is parsed accordingly. Our approach implementation resides just after the parsing step and the ingress pipeline (where the routing decision takes place), that is, in the egress pipeline. For this discussion, we assume that our forwarding device can process Ethernet frames and IP packets (or any known INT encapsulation protocol). Therefore, we omit the parsing steps since it is trivial and out of this work's scope. We then focused on the following up steps.

Our approach is implemented in the INT sink. Therefore, at this stage, all INT telemetry metadata has already been collected. The first steps of our approach consist of receiving the packet and extracting the telemetry data from it – or, in some cases, collecting them directly from the data plane (this might happen when the INT sink node acts as an INT transit node as well). This corresponds to steps 1 and 2 in Figure 3. The extracted data is then stored in a custom-made metadata header structure – named `ETA` metadata struct. This structure comprises M 32-bit structures and a 2-bit flag used to instruct the decision-making process. For instance, if a packet needs to be sent to the INT collector, this flag is used internally. In the Netronome architecture, the existing timestamps (`ingress_timestamp` and

`current_timestamp`) are 48-bit words. However, most of the applications use only the least significant bits (the last 32 bits) since they account for nanosecond differences. Further, as discussed next, the Netronome architecture also limits the results of arithmetic operations to 32 bits words (and therefore, we cannot operate on 48-bits timestamps). After extracting telemetry data into the `ETA` struct, our implementation calls a Micro-C extern code to handle more complex operations inside the data plane (depicted in Figure 4). For instance, floating-point operations are not allowed in the P4-16 language reference, nor does the SmartNIC natively implement it. To allow the SmartNIC code to be partially written in P4 and Micro-C – and more importantly, to exchange data between them – we rely on `ETA` structs and internal P4 metadata to enable such real-time communications.

The Micro-C code starts receiving data from the P4 pipeline and locking up memory regions using a mutex (steps 1-3 in Figure 4). Next, we calculate A_p and A_w according to Equation 1 and Equation 2, respectively. Figure 4 illustrates these procedures in steps 4-7. It is important to mention that neither the P4 reference architecture nor the Netronome support floating-point instructions. We implemented all these operations using fixed-point representation with 32 bits to surpass that limitation. In short, a fixed-point representation handles all real numbers as integers. The process scales up the numbers by multiplying by a constant factor C and then performs the multiplications/division as a regular integer. Finally, the scaled-up result is scaled down appropriately. In our implementation, we have used a constant C of 16 bits and performed the scale-up operation by applying a bit-shifting operation (i.e., $M_i' << 16$). After calculating A_p, we verify whether it is the case to send it out to the INT collector. We compare A_p with the observed A_e (which captures the historical behavior). In case the A_p packet value is higher than the dynamic A_e one, we mark the packet to be sent out to the collector (steps 8-12 in Figure 4).

As previously mentioned, our approach performs per-packet decisions. However, we update the exponential weighted moving average A_e at the end of a given window W. This is done because the average A_e depends on A_w – and, the Netronome architecture does not allow arbitrary integer divisions (only by the power of 2 by performing bit-shifting – for instance $M_i' >> 16$). Our approach verifies whether it reaches or not the end of a given window W (step 14 in Figure 4). If so, then it updates the moving average A_e (step 15-19 in Figure 4) according to Equation 3. Further, if it is the first time to reach the window W (i.e., $W' == W$), then A_e assumes a simple average of A_w (step 20 in Figure 4). Then, the Micro-C sets the mutex down and allows other threads to use the blocked shared memory. Last, our Micro-C code returns the calculated values to the original P4 pipeline (step 22-23 in Figure 4).

Back in the P4 pipeline (Figure 3), our approach clone and recirculate the packet p. The original packet is forwarded to its destination (step 13 in Figure 3), while the cloned one is sent to the INT collector (step 12 in Figure 3).

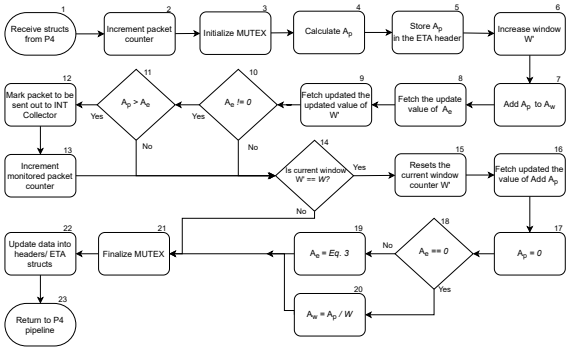


Fig. 4 Overview of the proposed Micro-C routine.

4 Performance Evaluation

In this section, we perform a performance evaluation of our proposal mechanism using a Netronome SmartNIC. We start by describing our environment setup, followed by discussing the results.

4.1 Environment Setup and Baseline Comparison

Setup. Our environment setup consists of three high-end servers. Each server has an Intel Xeon 4214R processor with 32 GB RAM. One server is our Device Under Test (DUT) – i.e., the server in which our solution (i.e., P4 program) is loaded – and the other two are used for traffic generation. All servers have a Netronome SmartNIC Agilio CX 10 Gbit/s network device with two network interfaces, which are physically connected (i.e., each traffic generator is connected to the DUT directly). We use MoonGen[10] as our DPDK[3] traffic generator. We instruct MoonGen using the Netronome Packet Generator[4]. In our experiments, we send 64B IPv4 packets at line rate (i.e., 10Gbit/s) with random source and destination prefixes. One of the traffic generator servers is used to send foreground network traffic, while the second is used to inject abrupt network traffic from time to time. This abrupt network traffic is generated with MoonGen to lead to a congested scenario. The experiment is run through 105 seconds, divided by time slots/epochs of 15 seconds each. We sent network traffic bursts in the following slots: $2nd$ (15s-30s), $4th$ (45s-60s), $6th$ (75s-90s). In our experiments, the SmartNIC Netronome acts simultaneously as an INT transit and an INT sink. First, it collects internal data plane metrics such as packet processing time and packet size (i.e., set M'). Then, we

[3] `https://www.dpdk.org/`

[4] `https://github.com/emmericp/MoonGen/tree/master/examples/netronome-packetgen`

assume that both metrics are weighted equally (i.e., $w_1 = w_2$) for calculating A_p, A_w, and A_e. We varied the window size W from 2^{20} to 2^{23} packets. Also, the parameters α in A_e varies from 0.1 to 0.9. Our solution is compiled using the Netronome's P4 compiler. The compiled code is statically assigned to a single micro engine (ME) inside the SmartNIC. This is done as there are some limitations on the Netronome's Mutex implementation when using different memories hierarchies (i.e., different from the ones inside the ME). All experiments were run at least 30 times to ensure a confidence level higher than 90%.

Baseline. We compare our `ETA` approach against the Full INT procedure and a fixed threshold one. In the former, all INT data are reported to the INT collector, while in the latter we use a constant value as the threshold. This constant value is obtained by calculating the average of all collected INT data in an offline manner. Our codes are publicly available[5] in order to foster reproducibility.

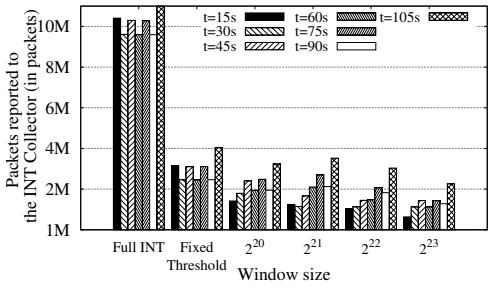


Fig. 5 Number of packets reported to an INT Collector.

4.2 Results

Number of packets sent to the INT Collector. We start by analyzing the number of packets that have been sent to an INT Collector in a given window W (Figure 5). For this experiment, we consider that $\alpha = 0.1$ (we later analyzed its impact). We observe that our approach outperforms the Full INT and Fixed-Threshold procedures in terms of packets reported by a factor of $16X$ and $5X$, respectively. The main reason consists of the dynamic adjustment made in the threshold value over time. Further, we also observe that our approach decreases the number of reported packets as the size of the windows W increases. For instance, we note 43% less network traffic being reported using a windows $W = 2^{23}$ than $W = 2^{20}$. With larger window size,

[5] urlhttps://github.com/canofre/mestrado/

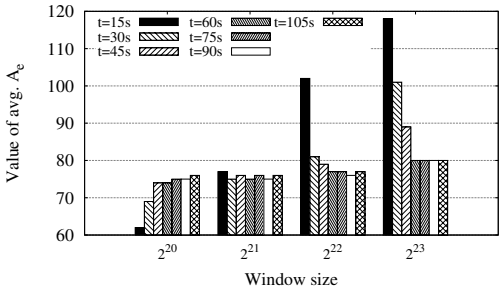


Fig. 6 Impact of window size on the average A_e with $\alpha = 0.2$

our metrics A_w tends to be smoother over time and less impacted by abrupt changes in the data plane metrics. Last, we also note a saw-tooth behavior between bars (e.g., $W = 2^{30}$), which represents the effect of sending packet bursts in a given time interval. The larger is the window size, the less is the saw-tooth behavior observed.

Impact of the window size on the computed average A_e. Next, we evaluate how the average A_e computed by the data plane evolves considering different window sizes (from 2^{20} to 2^{23}). Figure 6 illustrates the behavior when setting $\alpha = 0.2$. When the window size is small (e.g., 2^{20}), the average A_e increases over time until reaching a stable value. On the contrary, larger windows tend to decrease the average A_e value over time until reaching a similar stable value. With larger windows (i.e., 2^{22} and 2^{23}), the summation made before computing A_e within an epoch encompasses the periods where network bursts are sent. Therefore, it ends up increasing its initial value in comparison to small windows. Further, we also note that the value found in the stability tends to be more uniform in this case.

Impact of fine-tuning the parameters on the computed average A_e. We analyzed how the A_e evolves varying α from 0.1 to 0.9. For this experiment, we set the window size to $W = 2^{20}$. Note that higher values assigned to α tend to prioritize the behavior observed in the last time window (i.e., A_w), while lower values prioritize the historical behavior observed over time (i.e., A_e). As we can observe in Figure 7, the higher the α values the higher the average A_e gets over time. In this case, network traffic bursts change data plane metrics (e.g., in-network processing time) and then propagate such values' increases with more intensity to the following-up windows. In turn, lower values of α tend to prioritize the historical behavior and, therefore, the obtained values of A_e are less susceptible to short network traffic variations.

Impact on packet processing latency and throughput. Last, we evaluate the impact of our approach in terms of packet processing and latency when processing packets in the data plane. First, Figure 8(a) depicts the

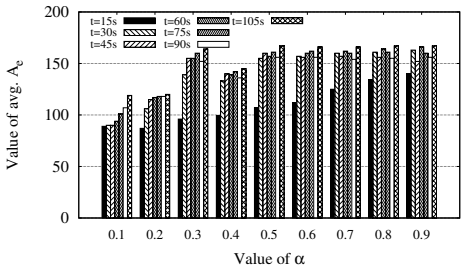
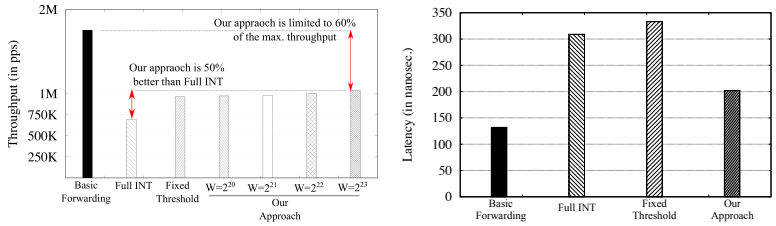


Fig. 7 Impact of the fine-tuning of α on the computed average A_e considering $W = 2^{20}$.



(a) Measured throughput. (b) Measured packet latency.

Fig. 8 Impact of Throughput and Latency of our proposed approach.

achieved throughput in packets per second. Our approach is able to outperform the Full INT strategy and the Fixed Threshold by 50% and by 10%, respectively. However, when compared to the Basic Forward – i.e., the same P4 code without any add-on – our approach is limited to run at most 60% of the maximum throughput. This occurs mostly because our approach demands more processing power and locks (due to mutex usage) and our approach is limited to running in a single ME (with 8 threads). In turn, Figure 8(b) illustrates the incurred data plane latency. We measured the latency as the difference between the ingress and egress timestamps inside the data plane. As we observe, our proposed approach adds around 60ns – that is, 1.53X higher than the Basic Forwarding approach. In turn, the Full INT and the Fixed Threshold approaches add 2.34X and 2.52X, respectively.

5 Related Work

The data plane programmability has opened up a wide range of research opportunities to solve existing network monitoring problems such as packet priorization [7], the usage of selective telemetry to reduce the network overload [14], and the classification and analysis of network flows [4]. As previously mentioned, INT allows the network flow packets to embed network teleme-

try data - such as in [26, 1, 6]. Sel-INT [26] leverages select group tables to selectively insert INT headers into software switches based on its bucket's weight and a certain probability. Similarly, PINT [1] leverages global hash functions and randomly decides to embed INT data in a given packet, while LINT [6] implements a mechanism with adaptive telemetry accuracy, where each node in the network analyzes its impact and decides whether to send the information to the collector.

P4Entropy [9] calculates the entropy of traffic by using the information contained in the packets. The entropy results are forwarded to the controller for storage and future analysis. Lin et al. [16] perform data collection to allow the SDN controller to evaluate the behavior of network traffic to improve data routing. Likewise, [8] proposes a DDoS schema entirely in the data plane. It leverages typical DDoS metrics such as incoming flows and packet symmetry ratio and periodically triggers alarms to external controllers.

With the emergency of hardware technologies such as SmartNICs and domain-specific programming languages such as P4, more applications are being implemented directly in the data plane. The implementation of selective and dynamic monitoring with the use of additional headers [25], the classification of packets into classes with the implementation of machine learning algorithms [29] and the detection of flow events [30] are some examples of programmatic packet header manipulation. NIDS [20] consists of a machine learning-based anomaly detection algorithm for detecting anomalous packets and future error mitigation. SwitchTree [15] performs the implementation of the Random Forest algorithm in the data plane for packet analysis and decision-making while updating the rules at runtime. [28] and [29, 15] present an implementation of machine learning models in the data plane using decision tree, and aim to generate network mapping for intrusion detection services.

Current research efforts in the INT domain (e.g., [1, 6, 26]) have mostly neglected the costs of reporting telemetry data to INT collectors. In fact, existing work have considered either a full report of information or a selective one based on static thresholds. In this work, we aim to fill this gap and offload the decision process of reporting INT data into the SmartNIC data plane in a dynamically manner. We propose a lightweight in-network mechanism based on a window-based moving average with collected INT data as input.

6 Final Remarks

In this paper, we propose a lightweight in-network mechanism to selective report in-band network telemetry to INT collectors. Our approach is based on a window-based moving average and it is tailored to SmartNIC architectures. By evaluating our approach in a state-of-the-art SmartNIC, we showed that our approach can report up to 16X less reporting statistics to INT collectors, while introducing a negligible overhead in terms of latency (1.5X higher than the baseline pipeline). As future work, we intend to explore machine learning

algorithms in the data plane to decide whether to report data or not. Also, we intend to explore other offloading alternatives so that the processing workload can be split up on different SmartNICs or in eBPF/DPDK approaches.

Acknowledgements

This work was partially funded by National Council for Scientific and Technological Development (CNPq) (grant 427814/2018-9), São Paulo Research Foundation (FAPESP) (grants 2018/23092-1, 2020/05115-4, 2020/05183-0), Rio Grande do Sul Research Foundation (FAPERGS) (grants 19/2551-0001266-7, 20/2551-000483-0, 19/2551-0001224-1, 21/2551-0000688-9).

References

1. Ben Basat, R., Ramanathan, S., Li, Y., Antichi, G., Yu, M., Mitzenmacher, M.: Pint: Probabilistic in-band network telemetry. In: Proceedings of ACM SIGCOMM. pp. 662–680 (2020)
2. Bhamare, D., Kassler, A., Vestin, J., Khoshkholghi, M.A., Taheri, J.: Intopt: In-band network telemetry optimization for nfv service chain monitoring. In: ICC 2019 - 2019 IEEE International Conference on Communications (ICC). pp. 1–7 (2019)
3. Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., Walker, D.: P4: Programming protocol-independent packet processors. ACM SIGCOMM 14 **44**(3), 87–95 (Jul 2014)
4. Castanheira, L., Parizotto, R., Filho, A.E.S.: Flowstalker: Comprehensive traffic flow monitoring on the data plane using P4. In: 2019 IEEE International Conference on Communications, ICC 2019. pp. 1–6. IEEE, Shanghai, China, May 20-24, 2019 (2019)
5. Castro, A.G., Lorenzon, A.F., Rossi, F.D., Da Costa Filho, R.I.T., Ramos, F.M.V., Rothenberg, C.E., Luizelli, M.C.: Near-optimal probing planning for in-band network telemetry. IEEE Communications Letters pp. 1–1 (2021)
6. Chowdhury, S.R., Boutaba, R., François, J.: Lint: Accuracy-adaptive and lightweight in-band network telemetry. In: IFIP/IEEE International Symposium on Integrated Network Management (IM). pp. 349–357 (2021)
7. Cugini, F., Gunning, P., Paolucci, F., Castoldi, P., Lord, A.: P4 in-band telemetry (int) for latency-aware vnf in metro networks. In: Optical Fiber Communication Conference (OFC) 2019. p. M3Z.6. Optical Society of America (2019)
8. Dimolianis, M., Pavlidis, A., Maglaris, V.: A multi-feature ddos detection schema on p4 network hardware. In: 2020 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN). pp. 1–6 (2020)
9. Ding, D., Savi, M., Siracusa, D.: Estimating logarithmic and exponential functions to track network traffic entropy in p4. In: NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium. p. 1–9. IEEE Press (2020)
10. Emmerich, P., Gallenmüller, S., Raumer, D., Wohlfart, F., Carle, G.: Moongen: A scriptable high-speed packet generator. In: Proceedings of the ACM IMC. p. 275–287. IMC '15, ACM, New York, NY, USA (2015)
11. Hohemberger, R., Castro, A.G., Vogt, F.G., Mansilha, R.B., Lorenzon, A.F., Rossi, F.D., Luizelli, M.C.: Orchestrating in-band data plane telemetry with machine learning. IEEE Communications Letters **23**(12), 2247–2251 (2019)
12. Jeyakumar, V., Alizadeh, M., Geng, Y., Kim, C., Mazières, D.: Millions of little minions: Using packets for low latency network programming and visibility. ACM SIGCOMM CCR **44**(4), 3–14 (2014)

13. Joshi, R., Qu, T., Chan, M.C., Leong, B., Loo, B.T.: Burstradar: Practical real-time microburst monitoring for datacenter networks. In: Proceedings of the 9th Asia-Pacific Workshop on Systems. APSys '18, Association for Computing Machinery, New York, NY, USA (2018)
14. Kim, Y., Suh, D., Pack, S.: Selective in-band network telemetry for overhead reduction. In: IEEE International Conference on Cloud Networking (CloudNet). pp. 1–3 (2018)
15. Lee, J.H., Singh, K.: Switchtree: In-network computing and traffic analyses with random forests. Neural Computing and Applications (11 2020)
16. Lin, W.H., Liu, W.X., Chen, G.F., Wu, S., Fu, J.J., Liang, X., Ling, S., Chen, Z.T.: Network telemetry by observing and recording on programmable data plane. In: IFIP Networking Conference (IFIP Networking). pp. 1–6 (2021)
17. Liu, Z., Bi, J., Zhou, Y., Wang, Y., Lin, Y.: Netvision: Towards network telemetry as a service. In: 2018 IEEE 26th International Conference on Network Protocols (ICNP). pp. 247–248 (Sep 2018)
18. Luizelli, M.C., Canofre, R., Lorenzon, A.F., Rossi, F.D., Cordeiro, W., Caicedo, O.M.: In-network neural networks: Challenges and opportunities for innovation. IEEE Network **35**(6), 68–74 (2021). https://doi.org/10.1109/MNET.101.2100098
19. Marques, J.A., Luizelli, M.C., Da Costa, R.I.T., Gaspary, L.P.: An optimization-based approach for efficient network monitoring using in-band network telemetry. Journal of Internet Services and Applications **10**(1), 16 (Jun 2019)
20. Nam, S., Lim, J., Yoo, J.H., Hong, J.W.K.: Network anomaly detection based on in-band network telemetry with rnn. In: IEEE International Conference on Consumer Electronics - Asia (ICCE-Asia). pp. 1–4 (2020)
21. Netronome: Internet (2020), `https://www.netronome.com/static/app/img/products/silicon-solutions/WP_NFP4000_TOO.pdf`
22. Pan, T., Song, E., Bian, Z., Lin, X., Peng, X., Zhang, J., Huang, T., Liu, B., Liu, Y.: Int-path: Towards optimal path planning for in-band network-wide telemetry. In: IEEE INFOCOM. pp. 1–9 (Apr 2019)
23. Saquetti, M., Canofre, R., Lorenzon, A.F., Rossi, F.D., Azambuja, J.R., Cordeiro, W., Luizelli, M.C.: Toward in-network intelligence: Running distributed artificial neural networks in the data plane. IEEE Communications Letters **25**(11), 3551–3555 (2021)
24. Singh, S.K., Rothenberg, C., Luizelli, M.C., Antichi, G., Pongracz, G.: Revisiting heavy-hitters: Don't count packets, compute flow inter-packet metrics in the data plane. In: ACM SIGCOMM Poster. pp. 1–4. ACM, New York, NY, USA (2020)
25. Suh, D., Jang, S., Han, S., Pack, S., Wang, X.: Flexible sampling-based in-band network telemetry in programmable data plane. ICT Express **6**(1), 62–65 (2020)
26. Tang, S., Li, D., Niu, B., Peng, J., Zhu, Z.: Sel-int: A runtime-programmable selective in-band network telemetry system. IEEE transactions on network and service management **17**(2), 708–721 (2019)
27. Viegas, P., Goes de Castro, A., Lorenzon, A.F., Rossi, F.D., Luizelli, M.C.: The actual cost of programmable smartnics: diving into the existing limits. In: Barolli, L., Amato, F., Moscato, F., Enokido, T., Takizawa, M. (eds.) Advanced Information Networking and Applications. pp. 381–392. Springer International Publishing (2021)
28. Xavier, B.M., Guimarães, R.S., Comarela, G., Martinello, M.: Programmable switches for in-networking classification. In: IEEE INFOCOM 2021 - IEEE Conference on Computer Communications. pp. 1–10 (2021)
29. Xiong, Z., Zilberman, N.: Do switches dream of machine learning? toward in-network classification. p. 25–33. HotNets '19, Association for Computing Machinery, New York, NY, USA (2019)
30. Zhou, Y., Sun, C., Liu, H.H., Miao, R., Bai, S., Li, B., Zheng, Z., Zhu, L., Shen, Z., Xi, Y., Zhang, P., Cai, D., Zhang, M., Xu, M.: Flow event telemetry on programmable data plane. In: Proceedings of ACM SIGCOMM. p. 76–89. SIGCOMM '20, Association for Computing Machinery, New York, NY, USA (2020)

Anexos

ANEXO A – CÓDIGO FONTE P4 DO APF

```
/* -*- APF - ANALISADOR PRELIMITAR DE FLUXO -*- */
#include <core.p4>
#include <v1model.p4>

/* Tipo de pacote - standard_metadata.instance_type */
#define PKT_NORMAL 0x0
#define PKT_CLONE_I2I 0x1
#define PKT_CLONE_E2I 0x2
#define PKT_CLONE_I2E 0x8
#define PKT_CLONE_E2E 0x9
#define PKT_RECIRCULADO 0x3

/* Divisor para controle de overflow */
#define DIVISOR 11    // 2048

const bit<16> TYPE_IPV4 = 0x800;
const bit<8> PROTO_UDP = 0x11;

typedef bit<32> var32_t;
typedef bit<48> var48_t;
typedef bit<64> var64_t;

register<var32_t>(4) reg32;

/*************************************************************
*************** H E A D E R S  ***************************
*************************************************************/
header ethernet_t {
  var48_t macDst;
  var48_t macSrc;
  bit<16>   etherType;
}

header ipv4_t {
  bit<4>  version;
  bit<4>  ihl;
  bit<8>  diffserv;
  bit<16> totalLen;
  bit<16> identification;
  bit<3>  flags;
```

```
41   bit<13> fragOffset;
42   bit<8>  ttl;
43   bit<8>  protocol;
44   bit<16> hdrChecksum;
45   var32_t srcAddr;
46   var32_t dstAddr;
47 }
48
49 header udp_t {
50   bit<16> srcPort;
51   bit<16> dstPort;
52   bit<16> lengthUdp;
53   bit<16> checksum;
54 }
55
56 header apf_t{
57   var32_t v1;
58   var32_t v2;
59   var32_t v3;
60   var32_t v4;
61   bit<2> analisar;
62 }
63
64 header intrinsic_metadata_t {
65   var64_t ingress_global_tstamp;
66   var64_t current_global_tstamp;
67   var32_t janela;   // tam janela
68   var32_t peso_mj;  // 1-lambda
69   var32_t peso_mh;  // lambda
70 }
71
72 struct metadata {
73   intrinsic_metadata_t intrinsic_metadata;
74 }
75
76 struct headers {
77   ethernet_t  ethernet;
78   ipv4_t      ipv4;
79   udp_t       udp;
80   apf_t       apf;
81 }
82
```

```
83 /*************************************************************
84 ************** P A R S E R  ****************************
85 *************************************************************/
86 parser MyParser(packet_in packet,
87         out headers hdr,
88         inout metadata meta,
89         inout standard_metadata_t smt) {
90
91   state start {
92     transition parse_ethernet;
93   }
94
95   state parse_ethernet {
96     packet.extract(hdr.ethernet);
97     transition select(hdr.ethernet.etherType){
98       TYPE_IPV4: parse_ipv4;
99       default: accept;
100    }
101  }
102
103  state parse_ipv4 {
104    packet.extract(hdr.ipv4);
105    transition parse_apf;
106  }
107
108  state parse_apf {
109    packet.extract(hdr.apf);
110    transition select(hdr.ipv4.protocol){
111      PROTO_UDP: parse_udp;
112      default: accept;
113    }
114  }
115
116  state parse_udp {
117    packet.extract(hdr.udp);
118    transition accept;
119  }
120 }
121
122 /*************************************************************
123 ******   C H E C K S U M  V E R I F I C A T I O N   ******
124 *************************************************************/
```

```
125 control MyVerifyChecksum(inout headers hdr, inout metadata meta)
       {
126   apply {  }
127 }
128
129 /*************************************************************
130 ******  I N G R E S S   P R O C E S S I N G *************
131 *************************************************************/
132 control MyIngress(inout headers hdr,
133           inout metadata meta,
134           inout standard_metadata_t smt) {
135   action drop() {
136     mark_to_drop();
137   }
138
139   action ipv4_forward(var48_t macDst, bit<16> port) {
140     smt.egress_spec = port;
141     hdr.ethernet.macSrc = hdr.ethernet.macDst;
142     hdr.ethernet.macDst = macDst;
143     hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
144   }
145
146   table tb_ipv4 {
147     key = {
148       hdr.ipv4.dstAddr: lpm;
149     }
150     actions = {
151       ipv4_forward;
152       drop;
153       NoAction;
154     }
155     size = 1024;
156     default_action = drop();
157   }
158
159   /* Inicializar as metricas a serem utilizadas no computo da
          medias */
160   action metricas_get(var32_t janela, var32_t peso_mh, var32_t
          peso_mj){
161     meta.intrinsic_metadata.janela = janela;
162     meta.intrinsic_metadata.peso_mh = peso_mh;
163     meta.intrinsic_metadata.peso_mj = peso_mj;
```

```
164     hdr.apf.analisar = 0;
165   }
166
167   table tb_metricas {
168     actions = {
169       metricas_get;
170       NoAction;
171     }
172     default_action = NoAction();
173   }
174
175   /* Alterar a porta de saida para interface v0_3 = 771 */
176   action monitor_send(bit<16> porta){
177     smt.egress_spec = porta;
178     hdr.ipv4.diffserv = 0xC8;
179   }
180
181   table tb_monitor {
182     actions = {
183       monitor_send;
184       NoAction;
185     }
186     default_action = NoAction();
187   }
188
189   /*
190   * Pacote valido e clonado, aplica a mudanca de porta para
        enviar
191   * ao coletor, caso contrario aplica o encaminhamento padrao
192   */
193   apply {
194     if ( hdr.ipv4.isValid() ){
195       if ( smt.instance_type ==  PKT_CLONE_E2I ){
196         tb_monitor.apply();
197       } else {
198         tb_ipv4.apply();
199         tb_metricas.apply();
200       }
201     }
202   }
203 }
204
```

```
205 /* Importacao do micro-c */
206 extern void analisaPacote();
207 /*************************************************************
208 ******  E G R E S S   P R O C E S S I N G   **************
209 *************************************************************/
210 control MyEgress(inout headers hdr,
211          inout metadata meta,
212          inout standard_metadata_t smt) {
213
214   /*
215   * Necessita de uma declaracao inicial externamente a action
       para
216   * recuperar o current_timestamp
217   */
218   var64_t ct64 = meta.intrinsic_metadata.current_global_tstamp;
219   var64_t it64 = meta.intrinsic_metadata.ingress_global_tstamp;
220   var64_t df64 = ct64-it64;
221
222   /*
223   * Calcula as metricas a serem utilizadas na avaliacao do pacote
224   * armazenando no cabecalho temporario do pacote
225   */
226   action metricas_get(){
227     hdr.apf.v1  = ct64[31:0]-it64[31:0];
228     hdr.apf.v2 = (var32_t)smt.packet_length;
229     hdr.apf.v3 = 0;
230     hdr.apf.v4 = 0;
231   }
232
233   /* Registra parciais dos valores de execucao */
234   action writeReg(){
235     ct64 = meta.intrinsic_metadata.current_global_tstamp;
236     hdr.apf.v4=(ct64[31:0]-it64[31:0]) >> DIVISOR;
237     reg32.write(0,hdr.apf.v1); // pacotes
238     reg32.write(1,hdr.apf.v2); // clonados
239     reg32.write(2,hdr.apf.v3); // mph
240     reg32.write(3,hdr.apf.v4); // latencia
241   }
242
243   apply {
244     if( smt.instance_type == PKT_NORMAL && hdr.ipv4.isValid() ){
245       metricas_get();
```

```
246
247         analisaPacote();
248
249         /* Clona os pacotes que excederem os parametros definidos
                */
250         if (hdr.apf.analisar == 1 ){
251           clone(CloneType.E2I,1);
252         }else{
253           /* Invalida o cabecalho dos pacotes nao clonados */
254           hdr.apf.setInvalid();
255         }
256
257         writeReg();
258
259       }else{
260         /* Invalida o cabecalho dos pacotes clonados */
261         hdr.apf.setInvalid();
262       }
263     }
264 }
265
266 /*************************************************************
267 *******   C H E C K S U M   C O M P U T A T I O N   *******
268 *************************************************************/
269 control MyComputeChecksum(inout headers hdr, inout metadata meta)
       {
270   apply {
271           update_checksum(
272         hdr.ipv4.isValid(),
273         { hdr.ipv4.version,
274           hdr.ipv4.ihl,
275           hdr.ipv4.diffserv,
276           hdr.ipv4.totalLen,
277           hdr.ipv4.identification,
278           hdr.ipv4.flags,
279           hdr.ipv4.fragOffset,
280           hdr.ipv4.ttl,
281           hdr.ipv4.protocol,
282           hdr.ipv4.srcAddr,
283           hdr.ipv4.dstAddr },
284          hdr.ipv4.hdrChecksum,
285          HashAlgorithm.csum16);
```

```
286   }
287 }
288
289 /*************************************************************
290 ************  D E P A R S E R  ***************************
291 *************************************************************/
292 control MyDeparser(packet_out packet, in headers hdr) {
293   apply {
294     packet.emit(hdr.ethernet);
295     packet.emit(hdr.ipv4);
296     packet.emit(hdr.udp);
297     packet.emit(hdr.apf);
298   }
299 }
300
301 /*************************************************************
302 ***************  S W I T C H  ***************************
303 *************************************************************/
304 V1Switch(
305   MyParser(),
306   MyVerifyChecksum(),
307   MyIngress(),
308   MyEgress(),
309   MyComputeChecksum(),
310   MyDeparser()
311 ) main;
```

ANEXO B – CÓDIGO FONTE DO MÓDULO EM MICRO-C COM MUTEX

```
1  /*
2  * Modulo externo com a utilizacao de mutex para controle de
3  * concorrencia, utilizando apenas uma das ilhas de FPCs.
4  */
5  #include <nfp.h>
6  #include <mutexlv.h>
7  #include <stdint.h>
8  #include <nfp/me.h>
9  #include <nfp/mem_atomic.h>
10 #include <pif_plugin.h>
11
12 /* Pesos para media_ponderada 2^16*/
13 #define PESO_TSTAMP 45875   // 0.7
14 #define PESO_PKTLEN 19660   //  0.3
15 #define SC_UP 16
16
17 /* Media calculada pela distribuicao de pacotes por intervalo */
18 #define MEDIA_FIXA 142
19 /* Divisor para controle de overflow */
20 #define DIVISOR 11       // 2048
21
22 /* Variaveis para registro */
23 __declspec(emem shared scope(global) export) uint32_t pacotes;
24 __declspec(emem shared scope(global) export) uint32_t clonados;
25
26 /* Variaveis ativas */
27 __declspec(emem shared scope(global) export) uint32_t janela;
28 __declspec(emem shared scope(global) export) uint32_t
      mp_acumulada;
29 __declspec(emem shared scope(global) export) uint32_t
      mp_historica=0;
30
31 /* Mutex */
32 typedef volatile __shared __gpr unsigned int MUTEXLV;
33 MUTEXLV lock=0;
34
35 int pif_plugin_analisaPacote(EXTRACTED_HEADERS_T *hdr,
      MATCH_DATA_T *meta){
36
```

```
37  PIF_PLUGIN_apf_T *apf = pif_plugin_hdr_get_apf(hdr);
38
39  __xwrite uint32_t xw = 0;
40  __xread uint32_t xr_mpa;
41  __xread uint32_t xr_mh;
42  __xread uint32_t xr_janela;
43  uint32_t mp = 0;
44
45  mem_incr32((__mem40 void*)&pacotes);
46
47  MUTEXLV_lock(lock,1);
48
49  /* Calcula media ponderada */
50  xw = mp = ((((uint64_t)PESO_TSTAMP*(uint64_t)apf->v1) /
51  (1 << SC_UP)) + ((PESO_PKTLEN*apf->v2) / 1 << SC_UP))) >>
       DIVISOR;
52
53  /* Incrementa janela: fora do mutex nao conta corretamente */
54  mem_incr32((__mem40 void*)&janela);
55  /* Recupera janela atual */
56  mem_read_atomic(&xr_janela,(__mem40 void*)&janela,sizeof(
       xr_janela));
57
58  /* Incrementa media acumulada */
59  mem_add32(&xw,(__mem40 void*)&mp_acumulada,sizeof(xw));
60
61  /* Recupera a media historica */
62  mem_read_atomic(&xr_mh,(__mem40 void*)&mp_historica,sizeof(
       xr_mh));
63
64  /* Compara media pacote com media historica e tamanho da janela
        */
65  if ( mp > xr_mh && xr_mh != 0 ){
66    apf->analisar=1;
67    mem_incr32((__mem40 void*)&clonados);
68  }
69
70  /* Recalcula a media historica */
71  if ( xr_janela == PKT_JANELA ){
72    /* Reseta contador da janela */
73    xw=0;
74    mem_write_atomic(&xw,(__mem40 void*)&janela,sizeof(xw));
```

```
75
76      /* Recupera media acumulada e reseta */
77      mem_read_atomic(&xr_mpa,(__mem40 void*)&mp_acumulada,sizeof(
            xr_mpa));
78      mem_write_atomic(&xw,(__mem40 void*)&mp_acumulada,sizeof(xw))
            ;
79
80      /* Calcula a nova media  historica - calculo 1 */
81      if (xr_mh != 0 ){
82        xw = ((xr_mpa / PKT_JANELA) * PESO_MJ) / (1 << SC_UP) +
83          (PESO_MH * xr_mh) / (1 << SC_UP);
84      }else{
85        /*  A primeira media  a media ponderada simples */
86        xw = (xr_mpa / PKT_JANELA );
87      }
88
89      mem_write_atomic(&xw,(__mem40 void*)&mp_historica,sizeof(xw))
            ;
90    }
91
92    MUTEXLV_unlock(lock,1);
93    apf->v1 = pacotes;
94    apf->v2 = clonados;
95    /* Recupera a media ponderada historica para registro */
96    mem_read_atomic(&xr_mh,(__mem40 void*)&mp_historica,sizeof(
          xr_mh));
97    apf->v3 = xr_mh;
98
99    return PIF_PLUGIN_RETURN_FORWARD;
100 }
101
102 int pif_plugin_analisaPacoteEstatico(EXTRACTED_HEADERS_T *hdr,
       MATCH_DATA_T *meta){
103
104   PIF_PLUGIN_apf_T *apf = pif_plugin_hdr_get_apf(hdr);
105   uint32_t mp = 0;
106
107   mem_incr32((__mem40 void*)&pacotes);
108
109   MUTEXLV_lock(lock,1);
110
111   /* Calcula media ponderada */
```

```
112   mp = ((((uint64_t)PESO_TSTAMP*(uint64_t)apf->v1) / (1 << SC_UP)
         ) +
113        ((PESO_PKTLEN*apf->v2) / (1 << SC_UP))) >> DIVISOR ;
114
115   /* Compara media pacote com media historica e tamanho da janela
          */
116   if ( mp > MEDIA_FIXA ){
117     mem_incr32((__mem40 void*)&clonados);
118     apf->analisar=1;
119   }
120
121   MUTEXLV_unlock(lock,1);
122   apf->v1 = pacotes;
123   apf->v2 = clonados;
124   return PIF_PLUGIN_RETURN_FORWARD;
125 }
126
127 /*
128 * Marca todos os pacotes para serem clonados. A realizacao deste
129 * processo com o micro-c facilita a contagem de pacotes
130 * por intervalo e assim o monitoramento do desempemho
131 */
132 int pif_plugin_intClassico(EXTRACTED_HEADERS_T *hdr, MATCH_DATA_T
       *meta){
133
134   PIF_PLUGIN_apf_T *apf = pif_plugin_hdr_get_apf(hdr);
135   uint32_t mp = 0;
136
137   mem_incr32((__mem40 void*)&pacotes);
138   apf->analisar=1;
139
140   apf->v1 = apf->v2 =  pacotes;
141
142   return PIF_PLUGIN_RETURN_FORWARD;
143 }
```

ANEXO C – CÓDIGO FONTE DO MÓDULO EM MICRO-C COM SEMÁFORO

```
1  /*
2  * Modulo externo com a utilizacao de semaforo vetorizado
3  * para controle de concorrencia nas as ilhas de FPCs.
4  */
5  #include <nfp.h>
6  #include <stdint.h>
7  #include <std/hash.h>
8  #include <nfp/me.h>
9  #include <nfp/mem_atomic.h>
10 #include "pif_plugin.h"
11
12 /*Pesos para media_ponderada 2^16*/
13 #define PESO_TSTAMP 45875   //  0.7
14 #define PESO_PKTLEN 19660   //  0.3
15 #define SC_UP 16
16
17 /* Divisor para controle de overflow */
18 #define DIVISOR 11       // 2048
19
20
21 /* Variaveis para registro */
22 __declspec(emem shared scope(global) export) uint32_t pacotes;
23 __declspec(emem shared scope(global) export) uint32_t clonados;
24
25 /* Variaveis ativas */
26 __declspec(emem shared scope(global) export) uint32_t janela[
      HASH_MAX+1];
27 __declspec(emem shared scope(global) export) uint32_t
      mp_acumulada[HASH_MAX+1];
28 __declspec(emem shared scope(global) export) uint32_t
      mp_historica[HASH_MAX+1];
29
30 /*  Inicializada em plugin_init_master */
31 __declspec(emem export aligned(64)) int global_semaforos[HASH_MAX
      +1];
32
33 /* Hash para obtencao do index do semaforo */
34 int getHash(uint32_t ip1, uint32_t ip2){
35   uint32_t hash_key[2];
```

```
36   uint32_t hash_id;
37   hash_key[0] = ip1;
38   hash_key[1] = ip2;
39   hash_id = hash_me_crc32((void *)hash_key,sizeof(hash_key), 1);
40   hash_id &= HASH_MAX;
41   return (int)hash_id;
42 }
43
44 void semaforo_down(volatile __declspec(mem addr40) void * addr) {
45   /* semaforo "DOWN" = claim = wait */
46   unsigned int addr_hi, addr_lo;
47     __declspec(read_write_reg) int xfer;
48   SIGNAL_PAIR my_signal_pair;
49
50   addr_hi = ((unsigned long long int)addr >> 8) & 0xff000000;
51   addr_lo = (unsigned long long int)addr & 0xffffffff;
52
53   do {
54      xfer = 1;
55      __asm {
56        mem[test_subsat, xfer, addr_hi, <<8, addr_lo, 1], \
57          sig_done[my_signal_pair];
58        ctx_arb[my_signal_pair]
59      }
60    } while (xfer == 0);
61 }
62
63 void semaforo_up(volatile __declspec(mem addr40) void * addr) {
64   /* semaforo "UP" = release = signal */
65   unsigned int addr_hi, addr_lo;
66   __declspec(read_write_reg) int xfer;
67   addr_hi = ((unsigned long long int)addr >> 8) & 0xff000000;
68   addr_lo = (unsigned long long int)addr & 0xffffffff;
69
70   __asm {
71     mem[incr, --, addr_hi, <<8, addr_lo, 1];
72   }
73 }
74
75 /*
76 * Chamada uma vez para todo sistema. Necessario habilitar
77 * opcao para compliacao via linha de comando
```

```
78 */
79 void pif_plugin_init_master() {
80   int i;
81   for (i = 0; i < HASH_MAX+1; i++) {
82     global_semaforos[i]=1;
83     semaforo_up(&global_semaforos[i]);
84   }
85 }
86
87 /* chamado uma vez para cada thread */
88 void pif_plugin_init() { }
89
90 int pif_plugin_analisaPacote(EXTRACTED_HEADERS_T *hdr,
      MATCH_DATA_T *meta){
91
92   PIF_PLUGIN_apf_T *apf = pif_plugin_hdr_get_apf(hdr);
93   PIF_PLUGIN_ipv4_T *ipv4 = pif_plugin_hdr_get_ipv4(hdr);
94
95   __xwrite uint32_t xw = 0;
96   __xread uint32_t xr_mpa;
97   __xread uint32_t xr_mh;
98   __xread uint32_t xr_janela;
99   uint32_t hash_id = 0;
100  uint32_t mp = 0;
101  int i=0;
102  mem_incr32((__mem40 void*)&pacotes);
103
104  hash_id = getHash(ipv4->srcAddr,ipv4->dstAddr);
105  semaforo_down(&global_semaforos[hash_id]);
106
107  /* Calcula media ponderada */
108  xw = mp = ((((uint64_t)PESO_TSTAMP*(uint64_t)apf->v1) /
109  (1 << SC_UP)) + ((PESO_PKTLEN*apf->v2) / 1 << SC_UP))) >>
       DIVISOR;
110
111  /* Incrementa janela: fora do mutex nao conta corretamente */
112  mem_incr32((__mem40 void*)&janela[hash_id]);
113  /* Recupera janela atual */
114  mem_read_atomic(&xr_janela,(__mem40 void*)&janela[hash_id],
       sizeof(xr_janela));
115
116  /* Incrementa media acumulada */
```

```
117   mem_add32(&xw,(__mem40 void*)&mp_acumulada[hash_id],sizeof(xw))
          ;
118
119   /* Recupera a media historica */
120   mem_read_atomic(&xr_mh,(__mem40 void*)&mp_historica[hash_id],
          sizeof(xr_mh));
121
122   /* Compara media pacote com media historica e tamanho da janela
          */
123   if ( mp > xr_mh && xr_mh != 0 ){
124     apf->analisar=1;
125     mem_incr32((__mem40 void*)&clonados);
126   }
127
128   /* Recalcula a media historica */
129   if ( xr_janela == PKT_JANELA ){
130     /* Reseta contador da janela */
131     xw=0;
132     mem_write_atomic(&xw,(__mem40 void*)&janela[hash_id],sizeof(
          xw));
133
134     /* Recupera media acumulada e reseta */
135     mem_read_atomic(&xr_mpa,(__mem40 void*)&mp_acumulada[hash_id
          ],sizeof(xr_mpa));
136     mem_write_atomic(&xw,(__mem40 void*)&mp_acumulada[hash_id],
          sizeof(xw));
137
138     /* Calcula a nova media  historica - calculo 1 */
139     if (xr_mh != 0 ){
140       xw = ((xr_mpa / PKT_JANELA) * PESO_MJ) / (1 << SC_UP) +
141         (PESO_MH * xr_mh) / (1 << SC_UP);
142     }else{
143       /*  A primeira media e a media ponderada simples */
144       xw = (xr_mpa / PKT_JANELA );
145     }
146
147     mem_write_atomic(&xw,(__mem40 void*)&mp_historica[hash_id],
          sizeof(xw));
148   }
149
150   semaforo_up(&global_semaforos[hash_id]);
151   apf->v1 = pacotes;
```

```
152   apf->v2 = clonados;
153   mem_read_atomic(&xr_mh,(__mem40 void*)&mp_historica[hash_id],
          sizeof(xr_mh));
154   apf->v3 = xr_mh;
155
156   return PIF_PLUGIN_RETURN_FORWARD;
157 }
158
159 /*
160 * Marca todos os pacotes para serem clonados. A realizacao deste
161 * processo com o micro-c facilita a contagem de pacotes
162 * por intervalo e assim o monitoramento do desempemho
163 */
164 int pif_plugin_intClassico(EXTRACTED_HEADERS_T *hdr, MATCH_DATA_T
        *meta){
165
166   PIF_PLUGIN_apf_T *apf = pif_plugin_hdr_get_apf(hdr);
167   uint32_t mp = 0;
168
169   mem_incr32((__mem40 void*)&pacotes);
170   apf->analisar=1;
171
172   apf->v1 = apf->v2 =  pacotes;
173
174   return PIF_PLUGIN_RETURN_FORWARD;
175 }
```

ANEXO D – TABELA DE CONSULTA

```
{
  "tables": {
    "ingress::tb_ipv4": {
      "rules": [
        {
          "name": "regra_1",
          "match": {"ipv4.dstAddr": {"value": "10.0.1.10"}},
          "action": {
            "type": "ingress::ipv4_forward",
            "data": {
              "macDst": {"value": "00:00:00:00:00:00"},
              "port": {"value": "p0"}
            }
          }
        },
        {
          "name": "regra_2",
          "match": {"ipv4.dstAddr": {"value": "10.0.2.10"}},
          "action": {
            "type": "ingress::ipv4_forward",
            "data": {
              "macDst": {"value": "00:00:00:00:00:00"},
              "port": {"value": "p1"}
            }
          }
        },
        {
          "name": "regra_3",
          "match": {"ipv4.dstAddr": {"value": "10.0.3.10"}},
          "action": {
            "type": "ingress::ipv4_forward",
            "data": {
              "macDst": {"value": "00:00:00:00:00:00"},
              "port": {"value": "p2"}
            }
          }
        },
        {
          "name": "regra_4",
          "match": {"ipv4.dstAddr": {"value": "10.0.4.10"}},
```

```
41            "action": {
42              "type": "ingress::ipv4_forward",
43              "data": {
44                "macDst": {"value": "00:00:00:00:00:00"},
45                "port": {"value": "p3"}
46              }
47            }
48          }
49        ],
50        "default_rule": {
51          "name": "drop",
52          "action": {"type": "ingress::drop"}
53        }
54      },
55      "ingress::tb_metricas": {
56        "default_rule": {
57          "name": "get_metricas",
58          "action" : {
59            "type": "ingress::metricas_get",
60            "data": {
61              "janela": {"value": "20"},
62              "peso_mh": {"value": "58982"},
63              "peso_mj": {"value": "6554"}
64            }
65          }
66        }
67      },
68      "ingress::tb_monitor": {
69        "default_rule": {
70          "name": "port_monitor",
71          "action" : {
72            "type": "ingress::monitor_send",
73            "data": {
74              "porta": {"value": "v0.2"}
75            }
76          }
77        }
78      }
79    }
80  }
```