UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Henrique Becker Brum

UM MÉTODO PARA COLETA DINÂMICA E EFICIENTE DE ESTATÍSTICAS EM REDES PROGRAMÁVEIS

Santa Maria, RS
2022

Henrique Becker Brum

UM MÉTODO PARA COLETA DINÂMICA E EFICIENTE DE ESTATÍSTICAS EM REDES PROGRAMÁVEIS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Ciência da Computação**.

ORIENTADOR: Prof. Carlos Raniery Paula dos Santos

509
Santa Maria, RS
2022

End. Eletr.: hbbrum@inf.com

Henrique Becker Brum

UM MÉTODO PARA COLETA DINÂMICA E EFICIENTE DE ESTATÍSTICAS EM REDES PROGRAMÁVEIS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Ciência da Computação**.

Carlos Raniery Paula dos Santos, Dr. (UFSM)
(Presidente/Orientador)

Raul Ceretta Nunes, Dr. (UFSM)

Roseclea Duarte Medina, Dr. (UFSM)

Santa Maria, RS
2022

RESUMO

UM MÉTODO PARA COLETA DINÂMICA E EFICIENTE DE ESTATÍSTICAS EM REDES PROGRAMÁVEIS

AUTOR: Henrique Becker Brum
ORIENTADOR: Carlos Raniery Paula dos Santos

O monitoramento de uma rede de computadores é uma atividade fundamental para que o funcionamento dela ocorra como desejado e sem problemas. Para isso é necessário que exista algum método de coleta para recuperar as informações da rede e enviá-las aos administradores dela. Além disso, é essencial que este método seja de fácil uso e eficiente para que ele mesmo não cause contratempos na rede. Sabendo da necessidade de monitorar uma rede e dos requisitos básicos para essa técnica, este trabalho tem como objetivo propor um método de coleta dinâmica e eficiente de estatísticas em redes programáveis. Dessa forma, foram investigados três tópicos diferentes: a linguagem de programação para processadores de pacotes P4, técnicas de telemetria em banda e algoritmos inteligentes para envio de estatísticas. Os resultados experimentais evidenciam que o método proposto é capaz de ajustar-se dinamicamente à diferentes padrões de tráfego para assim garantir uma alta eficiência de monitoramento. Por fim, conclui-se que a linguagem P4 é uma ferramenta ideal para utilizar no cenário apresentado.

Palavras-chave: Plano de dados programáveis. Coleta de Estatísticas Inteligente. Telemetria em banda. P4.

ABSTRACT

A METHOD FOR DYNAMIC AND EFFICIENT STATISTICS COLLECTION IN PROGRAMMABLE NETWORKS

AUTHOR: Henrique Becker Brum
ADVISOR: Carlos Raniery Paula dos Santos

Network monitoring is a fundamental activity for the correct and desired functioning of a computer network. In order to carry out this task, it is necessary that a method capable of retrieving network information and sending them to it's administrators exists. Furthermore, it's essential that this method is not only intuitive but also efficient, so no setbacks happen because of it. Knowing the need to monitor networks and the basic requirements for this goal, this work proposes a dynamic and efficient method for collecting statistics in programmable networks. To this end, three different subjects have been exhaustively studied: the P4 programming language, techniques for in-band telemetry and smart algorithms for statistics collection. The experimental results demonstrate that the proposed method is capable of dynamically adjusting itself to different traffic patterns in order to guarantee high monitoring efficiency. Besides that, the P4 language has shown itself to be an ideal tool for the presented scenario.

Keywords: Programmable Data Plane. Smart Statistics Collection. In-Band Telemetry. P4.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE ABREVIATURAS E SIGLAS

$P4$	Programming Protocol-Independent Packet Processors
SDN	Software-Defined Networking
PDP	Programmable Data Planes
$PISA$	Protocol Independent Switch Architecture
INT	In-band Telemetry
TCP	Transmission Control Protocol
MAC	Media Access Control
BGP	Border Gateway Protocol
IP	Internet Protocol
$IPv4$	Internet Protocol version 4
$IPv6$	Internet Protocol version 6
TDG	Table Dependency Graphs
API	Application Programming Interface
$REST$	Representational State Transfer
UDP	User Datagram Protocol
LPM	Longest Prefix Match
$RMSE$	Root Mean Square Error

SUMÁRIO

1 INTRODUÇÃO

As redes de computadores evoluíram drasticamente nas últimas décadas. O grande exemplo disso é a Internet, a qual começou como uma rede de uso acadêmico e militar e tornou-se a maior fonte global de informações, afetando imensamente a economia, política e cultura da sociedade moderna. O início da expansão global da Internet se deu na década de 80, quando o seus principais protocolos foram desenvolvidos, entre eles o *Internet Protocol* (IP) e o *Border Gateway Protocol* (BGP). Atualmente, esses protocolos ainda são utilizados como padrão em aplicações de rede, uma realidade preocupante visto que eles foram desenvolvidos nos primórdios da Internet e não são adaptados ao tamanho e à complexidade das modernas redes de computadores.

A ossificação da Internet é um problema resultante do enorme sucesso da Internet e de sua rápida aceitação global (TAYLOR, 2004). Com a ampla adoção dos protocolos desenvolvidos na década de 80, o núcleo da Internet acabou por consolidar-se com tecnologias incapazes de se adaptarem às necessidades contemporâneas. Assim, mesmo com o desenvolvimento de soluções inovadoras para uma vasta gama de problemas de redes pela comunidade acadêmica e profissionais da área, pouco avanço foi alcançado para implementar essas soluções em larga escala. Exemplo disso é o protocolo IPv6 (IETF, 1998) que mesmo tendo uma importância amplamente reconhecida demorou anos para chegar no estado de adoção atual. Outra área afetada é a de gerenciamento de redes, em que os administradores devem trabalhar com inúmeros eventos e aplicações utilizando ferramentas defasadas e de difícil uso, tornando essa tarefa lenta e propensa à erros.

Nos últimos anos, diversas novas tecnologias foram desenvolvidas para contornar as limitações apresentadas. Uma das mais interessantes e eficazes é a possibilidade de definir uma rede de computadores através de *software* (*Software-Defined Networking* - SDN). Esse paradigma surgiu em 2004 mas só ganhou notoriedade na área em 2008 com a criação do padrão *OpenFlow*, que disponibiliza um protocolo aberto para a programação das tabelas-fluxos de diferentes roteadores e *switches* (MCKEOWN et al., 2008). Mesmo apresentando várias vantagens em relação ao modelo tradicional, o *OpenFlow* possui um problema: ele suporta somente um conjunto limitado de protocolos e ações no seu ambiente. Sabendo dessa limitação, duas novas tecnologias foram desenvolvidas: primeiramente, chips programáveis para *switches* chamados de chips “PISA” (*Protocol Independent Switch Architecture*) e, em sequência, uma linguagem capaz de programá-los com o nome de P4. A partir do desenvolvimento dessas tecnologias tornou-se possível informar aos *switches* exatamente as ações que devem ser realizadas para o processamento dos pacotes (MCKEOWN, 2016).

Após o surgimento dessas novas técnicas, certas atividades realizadas no âmbito de Redes de Computadores tiveram que ser revisadas e atualizadas visto que essas novas

tecnologias permitem o desenvolvimento de aplicações que reconfiguram a rede automaticamente (TANGARI et al., 2018). Umas dessas tarefas é o de monitoramento da rede. Para esse fim surgiu a telemetria em banda (*In-band Network Telemetry* - INT), que é capaz de fornecer alta granularidade e medições em tempo real ao obter informações diretamente do plano de dados. Isso é possível através do uso dos pacote de dados para o transporte de cabeçalhos que serão preenchidos com estatísticas ao passarem por nós de trânsito da rede. Independentemente da metodologia de monitoramento, existem diversos trabalhos que investigaram maneiras de como abordar o *tradeoff* entre a precisão dos metadados coletados e o *overhead* causado pelo envio deles (*e.g.*, *Payless* (CHOWDHURY et al., 2014), *FlowSense* (YU et al., 2013), *sINT* (KIM; SUH; PACK, 2018)). Todavia, a vasta maioria das soluções encontradas usam o plano de controle para coordenar a requisição das estatísticas da rede e acabam por delegar ao plano de dados somente a função de processar os pacotes. Outro problema desses trabalhos, é a necessidade de possuir uma ideia do comportamento de um fluxo para configurar a solução.

Nesse contexto, o objetivo principal deste trabalho é desenvolver um método para envio dinâmico e eficiente de telemetria em redes programáveis. Através da linguagem de programação P4 é possível a definição do algoritmo de coleta de estatísticas diretamente no plano de dados, portanto, removendo os atrasos inerentes da comunicação com um controlador. E, paralelamente, o uso da telemetria em banda aumenta o desempenho da solução, já que os pacotes de dados da rede são usados também para o transporte das estatísticas.

As principais contribuições deste trabalho são: i) uso do plano de dados não somente para fazer o encaminhamento dos pacotes, mas também para controlar quando as estatísticas devem ser reportadas; ii) desenvolvimento de um algoritmo dinâmico de coleta de estatísticas que leva em conta o tráfego da rede para automaticamente se reconfigurar e melhorar a eficiência do processo de coleta; iii) utilização da técnica de telemetria em banda para envio de metadados da rede como método de monitoramento.

O restante deste trabalho está estruturado da seguinte maneira: o Capítulo 2 contém um referencial teórico sobre a programabilidade em rede, desde sua concepção até sua evolução para o P4, sobre técnicas de monitoramento, das clássicas até as que surgiram com o aparecimento da SDN, e, por fim, uma revisão de trabalhos relacionados. No Capitulo 3 é apresentado a solução desenvolvida, tanto o algoritmo principal da solução quanto detalhes da implementação. Em sequência, o Capítulo 4 contém a metodologia desenvolvida para os testes além dos resultados experimentais obtidos. E, finalmente, o Capítulo 5 apresenta as conclusões referentes ao trabalho e possíveis modificações futuras.

2 REFERENCIAL TEÓRICO

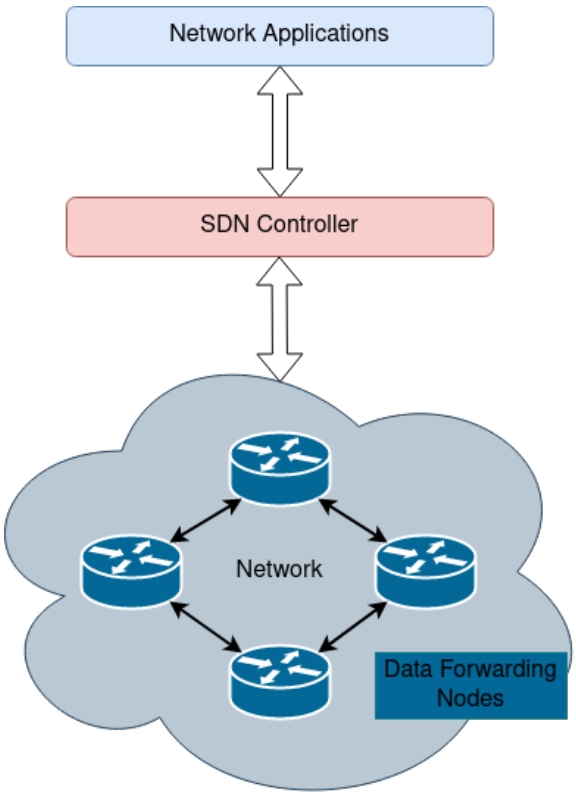
Neste capítulo será apresentado o material teórico por trás da solução desenvolvida além dos trabalho relacionados à ela. Primeiro será exposto a ideia geral das redes definidas por software (SDN), o padrão *OpenFlow*, que foi o ponto de início para a aceitação e propagação do uso de SDN, e a linguagem P4, a qual possibilitou a programação diretamente dentro do plano de dados. Em seguida, serão abordadas as diferentes técnicas existentes para o monitoramento de uma rede, desde as clássicas até as mais atuais como a telemetria em banda. Por fim, trabalhos relacionados serão revisados e discutidos.

2.1 SDN

Redes definidas por software ou *Software Defined Networking* é um paradigma que teve inicio em 2004 com o projeto *ForCES*, mas somente começou a ganhar mais importância e adeptos em 2008, com o desenvolvimento do *OpenFlow*. O SDN é baseado nas seguintes ideias: o plano de controle deve ser separado do plano de dados, as decisões de encaminhamento podem ser baseados em outros atributos além do endereço de destino, o controle da rede é logicamente centralizado e as funcionalidades da rede são programáveis através de aplicações rodando em conjunto com uma controladora. Mais especificamente, em uma rede definida por software, a funcionalidade do plano de dados de encaminhar pacotes é embutida dentro dos dispositivos de rede (*switches*, roteadores, etc), enquanto que a funcionalidade do plano de controle de gerenciar o comportamento desses aparelhos é colocado em um componente logicamente centralizado chamado controladora (CHOWDHURY et al., 2014). Além disso, existe a comunicação bilateral tanto entre as aplicações de rede e a controladora quanto entre a controladora e os dispositivos de rede.

Esse conjunto de ideais acabam por tornar SDN uma opção extremamente flexível para resolver problemas relacionados à rede de computadores, dado que para cada situação que ela for utilizada a rede pode ser reprogramada automaticamente para melhor satisfazer o objetivo desejado. Com isso, a atividade de monitoramento e configuração da rede realizada por administradores fica muito mais acessível e também é possível o fácil desenvolvimento de aplicações que necessitam de alto controle sobre o processamento de pacotes. A Figura 2.1 mostra uma versão simplificada de como é a arquitetura SDN, explicitando a separação entre o plano de controle e o de dados.

Figura 2.1 – Esquema do paradigma SDN



Fonte: Adaptado de Sarai (2019)

2.1.1 OpenFlow

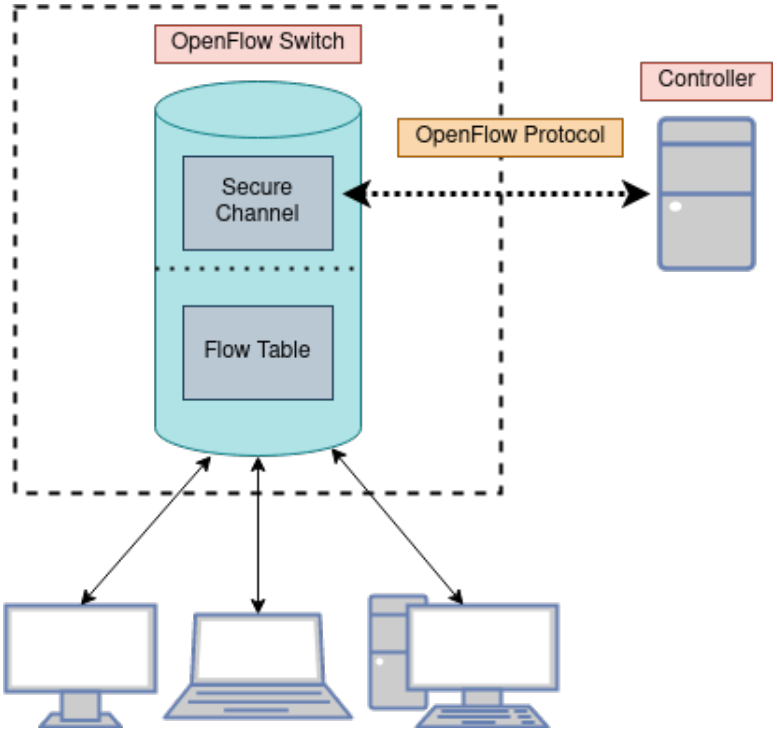
O padrão *OpenFlow* surgiu pela necessidade de padronizar a comunicação entre um plano de controle programável e vários dispositivos de rede (MCKEOWN, 2016). Para alcançar esse fim, o *OpenFlow* explora a existência de um grupo de funções que é comum entre *switches* e roteadores de diferentes vendedores (MCKEOWN et al., 2008). Por meio dessa particularidade foi criado um protocolo de comunicação chamado de *OpenFlow*, que possibilita um programa centralizado controlar, com certa limitação, o funcionamento dos aparelhos de rede independentemente do *hardware* por trás deles.

Em cada um dos *switches OpenFlow* existe uma tabela de fluxos onde as entradas dela representam um fluxo específico. Essas entradas consistem em três campos: um cabeçalho de pacote que determina o fluxo, uma ação de como os pacotes devem ser processados e as estatísticas do fluxo. A caracterização de que constitui um fluxo é ampla e só depende das capacidades de uma implementação particular da tabela-fluxo. Por exemplo, um fluxo poderia ser todos os pacotes que entram em certa porta do switch ou uma conexão TCP ou todos os pacotes que possuem certo endereço MAC. As ações associa-

das à cada fluxo também variam de acordo com a implementação, mas existem três ações básicas que todos *switches* devem disponibilizar: encaminhamento de pacotes para uma porta (ou portas), encapsulamento e envio dos pacotes de um fluxo à uma controladora e descartamento dos pacotes de um fluxo. Finalmente, algumas das estatísticas básicas suportadas são: contadores para o número de pacotes e *bytes* e o *timestamp* do ultimo pacote processado do fluxo.

A comunicação utilizando o protocolo *OpenFlow* funciona através da troca de mensagens entre a controladora e os *switches OpenFlow*. Por meio dessas mensagens, uma controladora pode adicionar, remover ou editar as entradas na tabela de fluxo do *switch* desejado. Além disso, ela também é capaz de solicitar estatísticas de um ou de vários fluxos ou até mesmo de automaticamente programar o *switch* para enviar essas informações com certa frequência. Na Figura 2.2 é apresentado o *switch OpenFlow* e o seu esquema de comunicação com os outros dispositivos da rede.

Figura 2.2 – Arquitetura do padrão *OpenFlow*



Fonte: Adaptado de (MCKEOWN et al., 2008)

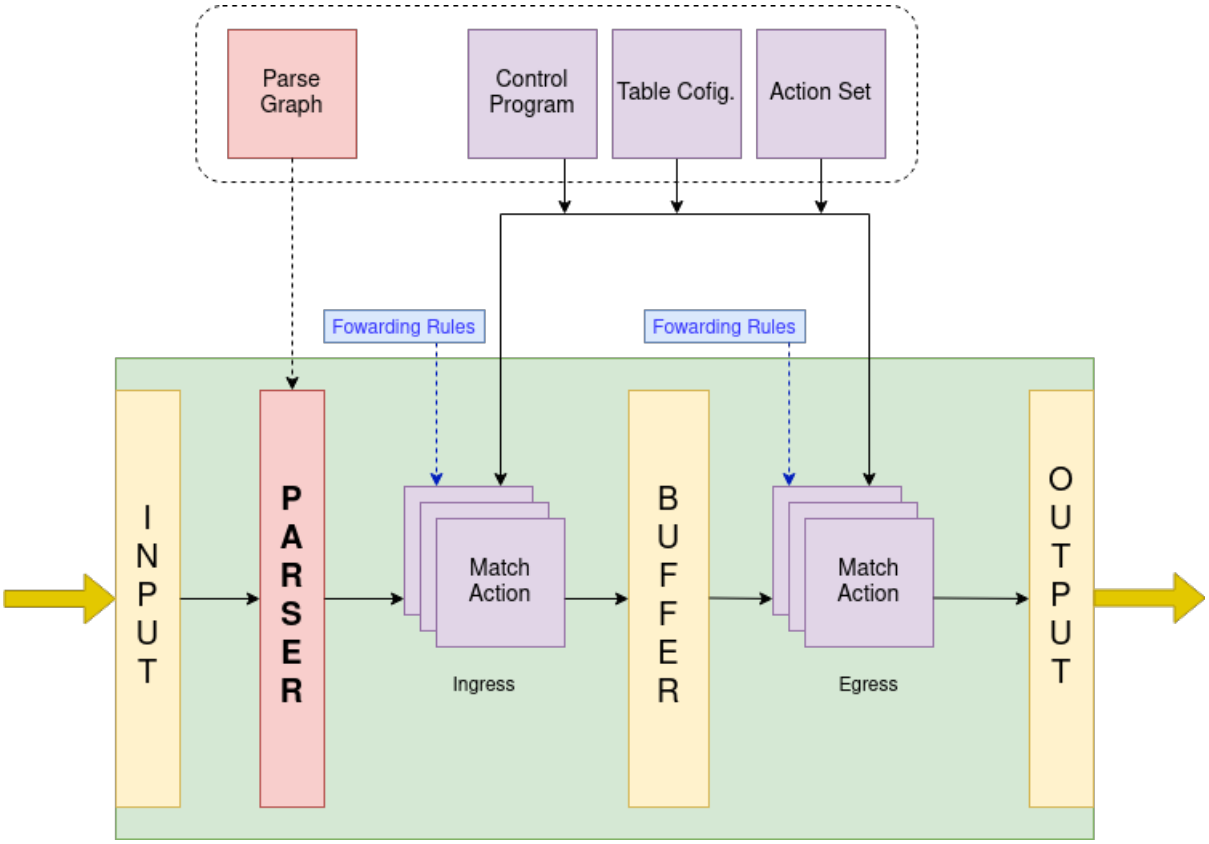
2.1.2 Linguagem P4

A linguagem P4 é uma linguagem de alto nível para a programação de processadores de pacotes independentemente do protocolo (BOSSHART et al., 2014). Os três pilares

dessa linguagem consistem da reconfigurabilidade, independência de protocolo e a independência de alvo. A reconfigurabilidade é a capacidade dos programadores em redefinir o processamento dos pacotes depois de instalados os dispositivos. A independência de protocolo garante que os dispositivos de rede não são vinculados a nenhum formato de pacote específico. E por último, a independência de alvo assegura que programas P4 não precisem ter conhecimento do *hardware* subjacente.

A Figura 2.3 mostra a abstração do modelo de encaminhamento de pacotes P4-14/PISA. Em um primeiro momento, os pacotes entram na fase de análise ou de *parser* em que seus cabeçalhos são extraídos. Após isso, os cabeçalhos são encaminhados para a etapa de *match+action*. Existem dois *pipelines* para a realização dessa parte: o de ingresso e de egresso. Em ambos os *pipelines* o funcionamento é praticamente o mesmo: o conteúdo dos cabeçalhos extraídos é comparado com as chamadas tabelas *match+action* para encontrar uma ação à ser executada nesses cabeçalhos. Durante os diferentes estágios, os pacotes podem carregar informações adicionais chamadas de metadados. Por fim, antes do pacote sair do switch, o pacote é remontado com os cabeçalhos modificados e está completo o processo de encaminhamento.

Figura 2.3 – Abstração do modelo de encaminhamento PISA



Fonte: Adaptado de (BOSSHART et al., 2014)

Os cabeçalhos ou *headers* descrevem a sequência e a estrutura de diferentes cam-

pos, como também o tamanho e a restrições de cada um deles (BOSSHART et al., 2014). A linguagem P4 permite uma definição de cabeçalhos bem ampla, desde *headers* que representam os protocolos mais comuns (*Ethernet*, IPv4, TCP, etc.) até protocolos totalmente customizados pelo programador. Além dos cabeçalhos, também existem os chamados metadados (*metadata*), que são estruturas para o transporte de informações adicionais durante o trânsito do pacote. Existem dois tipos de metadados: os definidos pelo usuário e os fornecidos pela arquitetura P4, chamados de metadados intrínsecos. Na Figura 2.4 pode ser visto um exemplo de um cabeçalho que representa o protocolo *Ethernet*, um protocolo de autoria própria chamado "MyProtocol_h" e alguns metadados definidos pelo usuário.

Figura 2.4 – Exemplos de cabeçalhos e metadados

```
1  header Ethernet_h {
2      bit<48> dstAddr;
3      bit<48> srcAddr;
4      bit<16> etherType;
5  }
6
7  header MyProtocol_h {
8      bit<8> nextHeader;
9      bit<32> switchID;
10     bit<32> flowID;
11     bit<48> lastMatch;
12 }
13
14 struct metadata{
15     bit<8> egressSpec;
16     bit<8> auxValue;
17 }
```

Fonte: Autor

A primeira etapa do processamento do pacote é a de análise. Nela serão extraídos os campos dos cabeçalhos desejados. O P4 define essa etapa como uma máquina de estados finita que possui um estado inicial e um final como também estados intermediários, estes são implementados pelo programador. A maneira de transição entre os estados intermediários ocorre normalmente através de certos campos extraídos de pacotes anteriores. Esse comportamento é detalhado na Figura 2.5, onde o campo do protocolo *Ethernet* e do "MyProtocol" são utilizados para escolher o próximo cabeçalho ou estado de análise (linhas 7 e 17). Ao final desse processo existem duas possíveis situações, o de aceitação ou de rejeitamento dos cabeçalhos. Se forem aceitos, os *headers* extraídos serão encaminhados à

etapa de *match+action* do *pipeline* de ingresso. Caso rejeitados fica como implementação de cada arquitetura, mas o mais comum é simplesmente descartar o pacote.

Figura 2.5 – Etapa de análise da linguagem P4

```
state start {
    transition parse_Ethernet;
}

state parse_Ethernet {
    packet.extract(hdr.Ethernet);
    transition select(hdr.Ethernet.etherType) {
        0x1000: parse_MyProtocol;
        0X800: parse_IPV4;
        default: accept;
    }
}

state parse_MyProtocol {
    packet.extract(hdr.MyProtocol);
    transition select(hdr.MyProtocol.nextHeader) {
        0X800: parse_IPV4;
        default: accept;
    }
}
```

Fonte: Autor

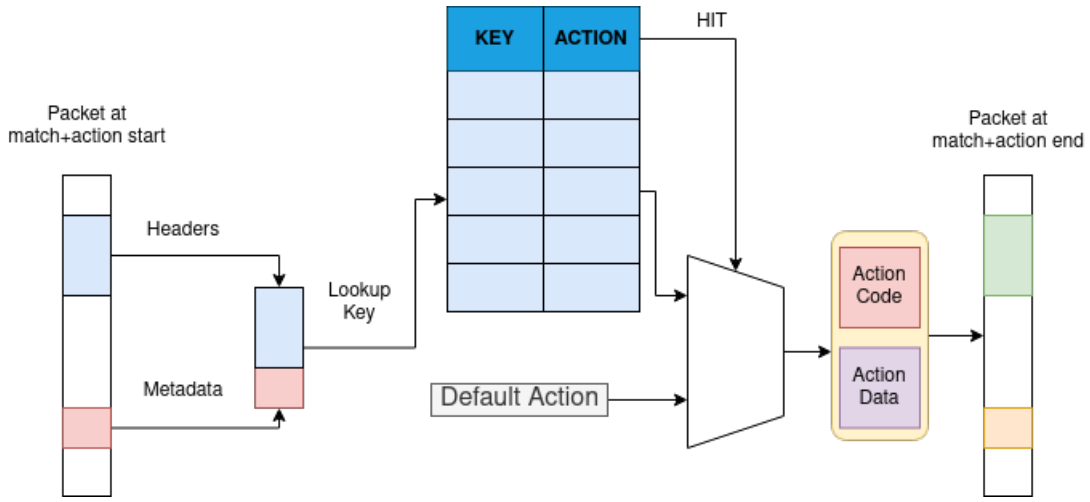
A próxima fase do processamento de pacotes do P4 é o *pipeline* de *match+action*. O propósito dessa parte é a comparação de um ou mais campos dos cabeçalhos extraídos no *parser* com uma tabela programada pelo controlador. Caso ocorra uma correspondência equivalente na tabela, uma ação associada à ela é executada. As ações geralmente consistem em modificar certos campos dos cabeçalho, como a atualização do endereço MAC de destino, ou dos metadados do programa.

As tabelas são estruturas que contém as instruções para o encaminhamento dos pacotes. Elas são compostas por chaves e um grupo de ações correspondentes. As chaves servem para especificar quais são os valores do plano de dados que vão ser utilizados na parte de comparação com os *headers*. Mais especificamente, cada chave é composta de uma lista de pares da forma *(e:m)*, onde o *e* é uma expressão dos valores utilizada na comparação enquanto que o *m* é o algoritmo que define como esses dados serão correspondidos.

As ações são pedaços de código capazes de manipular os cabeçalhos e meta-

dados de um pacote. É por meio delas que o plano de controle pode dinamicamente influenciar o funcionamento do plano de dados. Sintaticamente elas são semelhantes à funções sem retorno e a na sua composição só são permitidos primitivas simples. No decorrer do processo de *match+action* caso ocorra um *match*, os dados necessários para as ações são informados pelo plano de controle. Na Figura 2.6 o comportamento do estágio *match+action* pode ser compreendido com mais clareza.

Figura 2.6 – Fluxo do processo de *match+action*

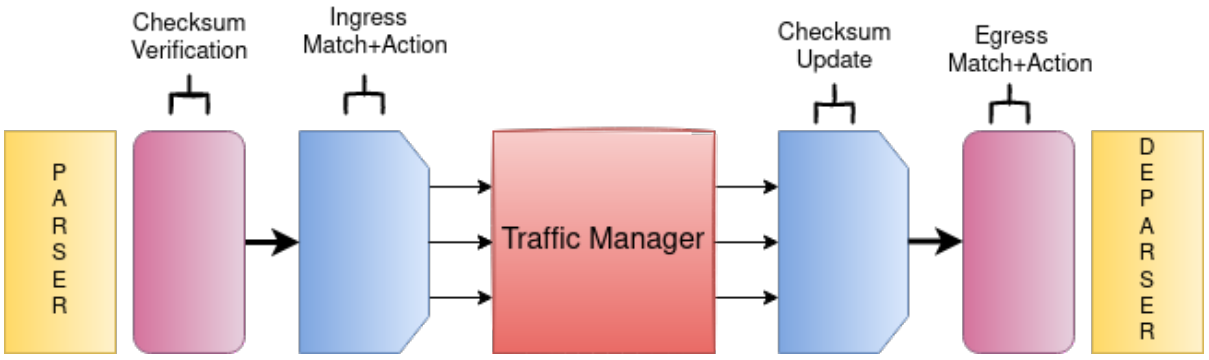


Fonte: Adaptado de (P4-CONSORTIUM, 2017)

Com o crescimento da linguagem e surgimento de outras arquiteturas além da PISA, foi necessário que o *pipeline* de processamento do P4 se tornasse flexível. Por causa disso criou-se o chamado modelo de arquitetura P4. Nesse modelo, o objetivo é identificar os blocos operacionais presentes em um determinado alvo (hardware subjacente) e especificar uma interface para programá-los. Os blocos operacionais são os blocos de análise e os de controle. Os blocos de análise servem para extrair headers ao passo que nos blocos de controle é realizado o processamento dos campos extraídos e o processo de *match+action*. Além de definir esses componentes, cada arquitetura também pode adicionar novos tipos de dados, constantes, objetos externos, etc.

A arquitetura padrão para a maioria dos programas P4 e também utilizada nesse trabalho é a *Simple Switch Architecture* ou *v1model*. Nela existem seis blocos operacionais, um de análise e cinco de controle. O primeiro bloco é o de análise (*parser*), e é nele que é feito a extração dos cabeçalhos. Logo em seguida vem uma seção de verificação do *checksum* do pacote. Em sequência existe dois blocos de controle chamados de bloco de ingresso e de egresso, onde são utilizado as tabelas de encaminhamento em conjunto com os cabeçalhos para realizar o *match+action*. Após esses blocos, existe um unidade de atualização do *checksum*, já que campos do cabeçalho podem ter sido modificados. Por fim, uma seção de controle (*deparser*) reinsere os cabeçalhos extraídos no *parser* de volta ao pacote. A Figura 2.7 mostra os blocos operacionais do *v1model* e a organização deles.

Figura 2.7 – Estrutura da arquitetura *v1model*



Fonte: Autor

Ademais dos blocos operacionais, a arquitetura *v1model* define vários metadados e objetos externos. Os metadados são bem variados, contendo desde *timestamps* sobre a entrada e saída dos pacote até informações sobre as fila das portas de saída do *switch*. Os objetos externos também são diversos e incluem tanto novas estruturas de dados quanto funções. Exemplos de novas estruturas de dados são os registradores que por serem memórias de propósito geral possuem a capacidade de guardar informações anteriores ao pacote atual. Já um exemplo de funções, é o método *recirculate*, no qual um pacote recomeça o *pipeline* de processamento após o bloco de egresso.

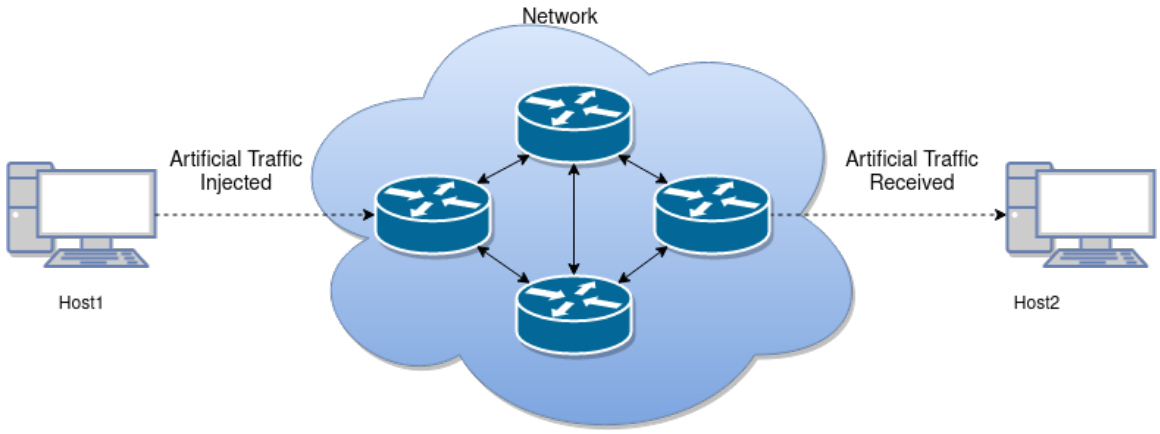
Com a arquitetura e o programa P4 definidos, a última etapa a ser realizada é a compilação deles e a geração da interface de comunicação entre o plano de controle e o de dados. O compilador P4 ou *p4c* realiza isso em dois estágios. Em um primeiro momento, o programa de controle P4 é convertido para uma representação gráfica das dependências entre as tabelas, conhecida como TDG (Table Dependency Graphs). Posteriormente, um *back-end* único de cada *switch* mapeia essa representação aos recursos específicos do dispositivo. E assim, está tudo concluído para o funcionamento da aplicação desenvolvida.

2.2 MONITORAMENTO DA REDE

O monitoramento da rede é fundamental ao sucesso das operações de rede, especialmente para a atual proporção que elas alcançaram. Para acompanhar a evolução das redes, as técnicas de monitoramento tiveram que se adaptar à elas. De acordo com Tan et al. (2020), essas técnicas podem ser divididas em três fases de pesquisa: i) métodos clássicos desenvolvidos desde 1995 quando a *Natural Science Foundation* dos Estados Unidos começou a fundar pesquisas sobre medição da Internet; ii) monitoramento definido por software desenvolvido e produzido por redes definidas por software desde 2008; iii) telemetria de rede com a ascensão dos planos de dados programáveis (PDP) desde 2015.

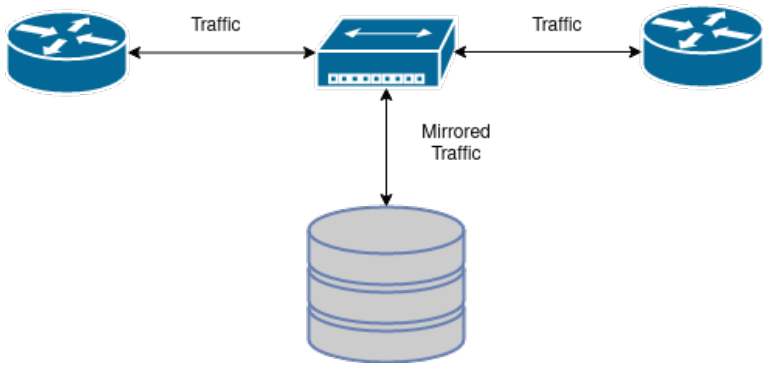
Os métodos tradicionais de monitoramento de tráfego também são divididos em três categorias: ativo, passivo e híbrido. A ideia por trás do monitoramento ativo é enviar pacotes de testes (*probes*) de um computador à outro e observar o comportamento desse tráfego à medida que ele atravessa a rede. Por utilizar pacotes gerados artificialmente os resultados desse procedimento não refletem 100% a situação real da rede. Diferentemente do ativo, o monitoramento passivo utiliza o tráfego real da rede para descobrir o status dela. Consequência disso, é a necessidade de hardware especializado, o que por sua vez limita a qualidade do monitoramento à performance do dispositivo. Por fim, o método hibrido busca encontrar soluções melhores combinando o método ativo e passivo ou reestruturando o mecanismo de monitoramento com as vantagens das duas outras técnicas. A Figura 2.8 ilustra o funcionamento do método ativo enquanto que a Figura 2.9 expõe o monitoramento passivo.

Figura 2.8 – Monitoramento ativo



Fonte: Adaptado de (NAVEEN; KUMAR; KOUNTE, 2013)

Figura 2.9 – Monitoramento passivo



Fonte: Adaptado de (NAVEEN; KUMAR; KOUNTE, 2013)

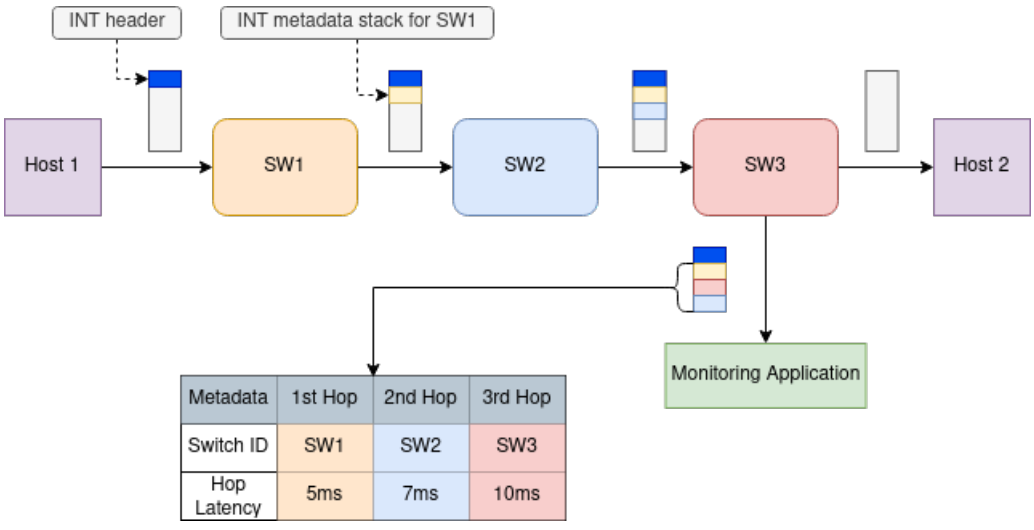
Com o surgimento do modelo SDN e do PDP, foi possível reconstruir o plano de dados e de controle da rede sem modificar a fundação da rede IP. Por causa disso, a

arquitetura de monitoramento da rede ficou muito mais flexível ao mesmo tempo que os algoritmos de coleta e de amostragem tradicionais foram preservados. No paradigma SDN, os *switches* são capazes de gerar estatísticas por fluxo, como a contagem de *bytes* e pacotes e ou a duração do fluxo. Existem duas maneiras de uma controladora SDN coletar estatísticas dos *switches* subjacentes: uma é baseada em *push* e a outra baseado em *pull*. Na abordagem *push*, a controladora recebe passivamente relatórios dos dispositivos, enquanto que na *pull*, a controladora solicita as informações aos nós da rede através de mensagens de controle.

Também aproveitando as capacidade de *switches* SDNs de gerarem informações localmente, foi proposto a ideia de telemetria de rede. Esse conceito diz respeito ao processo automatizado de remotamente coletar e processar informações da rede. Ao combinar automatização com SDNs, a telemetria acabou por se tornar uma opção adequada para os problemas de visibilidade de rede com escalabilidade e precisão, que os métodos clássicos de monitoramento vem enfrentando (GOMEZ; SHAMI; WANG, 2021).

In-band telemetry (INT) ou telemetria em banda clássica é um método emergente de telemetria onde o estado da rede é coletado através da inserção de metadados em pacotes ao passarem pelos nós da rede. Nesse sentido, dados de telemetria e do usuário compartilham a mesma conexão e o mesmo pacote. Devido à isso, a técnica INT é capaz de disponibilizar informações em tempo real de alta qualidade para as aplicações de monitoramento. Por outro lado, aplicações baseadas em telemetria precisam utilizar algoritmos de coleta de informações que sejam eficientes, visto que a inserção de informações da rede em cada pacote contribui para uma degradação no desempenho das aplicações e serviços utilizados pelos usuários finais (MARQUES et al., 2019). A Figura 2.10 mostra um exemplo de um método INT, em que um fluxo fluindo do *host* 1 ao *host* 2 passa pelos dispositivos de rede e em cada um deles a latência de *hop* é coletada.

Figura 2.10 – Exemplo de telemetria em banda



Fonte: Adaptado de (KIM; SUH; PACK, 2018)

2.3 TRABALHOS RELACIONADOS

O trabalho de Yu et al. (2013) apresenta um método de medição do tráfego de SDN através da abordagem baseada em *push* onde custo de monitoramento é zero. Para isso, os autores aproveitaram que mensagens de controle enviadas dos *switches* à controladora carregam informações que permitem estimar a performance da rede.

Em redes *OpenFlow* existem duas mensagens básicas que os dispositivos informam à controladora sobre um fluxo: *PacketIn* e *FlowRemoved*. A mensagem *PacketIn* informa quando um fluxo chega no *switch* enquanto que a *FlowRemoved* indica quando a entrada de um fluxo expira e informações sobre ele, como a duração e a quantidade de *bytes*. A partir disso, o algoritmo de proposto pelos autores usa o aviso *PacketIn* para inferir quais fluxos estão ativos em um determinado tempo e a mensagem *FlowRemoved* para determinar o final da existência do fluxo e também a utilização de *link*desse fluxo.

Empregando um pequeno banco de ensaio *OpenFlow* e simulações com o tráfego da rede de um campus, os autores mostraram que o método é capaz de prover informações atualizadas e precisas se a quantidade de mensagens de controle for abundante.

Chowdhury et al. (2014) propõe um *framework* de monitoramento de baixo custo para SDN. Focando no *tradeoff* entre precisão de monitoramento e saturação da rede, os autores desenvolveram uma API RESTful flexível para a coleta de estatísticas de fluxo em diferentes níveis de agregação. Essa solução usa um algoritmo adaptativo de coleta de estatísticas que entrega informações em tempo real altamente precisas sem implicar aumento significativo de sobrecarga na rede.

O *framework* possui cinco componentes principais: interpretador de requisição, o planejador, o selecionador de *switch*, o agregador e um banco de dados. O interpretador é responsável por traduzir as mensagens de alto nível da API de monitoramento para primitivas no nível de fluxos. O planejador agenda quando que os dados de um dispositivo devem ser coletados. O selecionador se encarrega de escolher qual *switch* ou *switches* devem enviar as informações à controladora. E por fim, o agregador agrupa os dados coletados de acordo com o nível de inclusão informado (*e.g* fluxo, tabela, porta, etc.) e guarda eles no banco de dados.

O artigo também propõe um algoritmo de monitoramento dinâmico para coletar as estatísticas eficientemente e sem adicionar sobrecarga na rede. Nesse algoritmo, a ideia principal é atualizar o tempo em que as estatísticas são coletadas com base na diferença entre o valor atual de *bytes* com o valor reportado na última vez. Mais especificamente, caso a diferença de *bytes* seja menor que um valor esperado, o tempo de coleta é aumentado, enquanto que se a diferença for maior que outra quantia fornecida, o intervalo é reduzido.

Com intuito de demonstrar a eficácia da solução proposta foi feito a avaliação e a comparação com o trabalho de Yu et al. (2013) e um método periódico de coleta. Os

resultados corroboram a eficiência do algoritmo desenvolvido, visto que a precisão das estatísticas coletadas foi maior que as do trabalho de Yu et al. (2013), enquanto que a a sobrecarga de rede só é 50% da sobrecarga do esquema periódico.

Kim, Suh e Pack (2018) buscam solucionar dois problemas comuns em aplicações de monitoramento que usam o *framework* INT (P4-CONSORTIUM, 2020): a sobrecarga da rede e a saturação do mecanismo de monitoramento existente. Esses problemas acontecem por causa do INT monitorar todos os pacotes da rede e consequentemente inserir em todos eles o cabeçalho INT e os metadados de cada *switch* percorrido. Para resolver essas adversidades, os autores apresentam um sistema capaz de ajustar a razão com que pacotes serão escolhidos para carregar o *header* INT com base na frequência de mudanças significativas nas informações da rede.

O modelo desenvolvido introduz dois novos componentes ao *framework* INT, uma tabela no *host* de origem INT e um algoritmo de reajuste da taxa de inserção no mecanismo de monitoramento.

A tabela serve simplesmente para mapear o ID do fluxo, que é um *hash* da quíntupla (IP de origem, porta de origem, IP de destino, porta de destino e tipo de protocolo), com sua taxa de inserção correspondente. Antes do pacote sair do *host* é feito o cálculo da *hash* do fluxo e comparado com a tabela. Caso encontre uma entrada, um *header* INT é inserido ou não no pacote dependendo da razão de inserção retornada.

Já o algoritmo proposto tem como objetivo tornar dinâmica a taxa de inserção de cada fluxo. Ele funciona da seguinte maneira: ao receber o *n*-ésimo pacote de telemetria do fluxo *f* é feita uma comparação entre os valores dos metadados recebidos com os do pacote *n-1* por uma função discriminante. Dessa função será retornado um valor binário o qual indica se houve ou não mudança significativa entre os dois pacotes. Se houver mudança, a taxa de inserção é aumentada. Caso não tenha acontecido um alteração expressiva durante o tempo de estabilização, a razão é configurada para um valor mínimo pré-definido.

Os resultados preliminares comprovam que o método implementado é capaz de mitigar a saturação da rede, reduzindo em até 37% a sobrecarga causada pelos metadados coletados em relação ao INT sem esse algoritmo.

2.4 DISCUSSÃO

Em Yu et al. (2013), o mecanismo de medição do tráfego de um fluxo é feito somente com mensagens passivas vindas do *switch*. Enquanto que a sobrecarga de rede é extremamente baixa, a sua estimativa da utilização do *link* é bem distante do valor real e só funciona mais precisamente quando os fluxos da rede são de baixa duração. E mesmo com fluxos curtos, esse método não seria capaz de detectar variações pontuais na vazão

de um fluxo, visto que ele só retorna a utilização média do fluxo durante a sua existência.

Para evitar as limitações do método de Yu et al. (2013), este trabalho envia ativamente as estatísticas dos *switches* e assim aumenta a precisão do conhecimento da rede. Além disso, ele também adapta o tempo em que as informações do tráfego são enviadas, para que seja possível a captura de variações na vazão de dados.

Em Chowdhury et al. (2014) é apresentado um *framework* de monitoramento em redes definidas por software com baixa sobrecarga da rede e alta precisão das informações. Entretanto, existem algumas limitações desse artigo. A primeira é que ele utiliza do plano de controle para enviar as mensagens de coleta de informações, gerando custo extra para fazer a coleta. E a segunda, é de que o algoritmo de extração de estatísticas necessita de uma ideia sobre a quantidade de dados que irão passar na rede, para assim determinar se aumenta ou diminui o período de coleta.

Diferentemente dos trabalhos anteriores que necessitam do envio de mensagens entre uma controladora e os*switches* para a coleta de estatísticas, os pesquisadores Kim, Suh e Pack (2018) desenvolveram uma solução que não necessita dessa comunicação pois utiliza a telemetria em banda, assim como proposto neste trabalho. Porém, a estratégia empregada por eles ainda utiliza uma controladora para atualizar a tabela das taxas de inserção no dispositivo de origem do fluxo. Além disso, é importante notar que o algoritmo de atualização da frequência com que o *header* INT é adicionado à um pacote necessita de valores específicos da rede para determinar a nova periodicidade.

Neste trabalho, com intenção de acabar com a comunicação da aplicação de monitoramento com os dispositivos de encaminhamento de pacotes e assim minimizar o custo extra causado pelas mensagens do plano de controle, o algoritmo de coleta é programado diretamente no plano de dados através da linguagem P4 e o envio das estatísticas é feito por telemetria em banda. Ademais, o algoritmo de coleta implementado não necessita de muito conhecimento prévio sobre a rede para funcionar como esperado o que facilita o trabalho de monitoramento .

3 SOLUÇÃO PROPOSTA

Este capítulo tem como propósito apresentar a lógica dos algoritmos desenvolvidos para a solução, como também detalhes da implementação deles e do trabalho como um todo.

3.1 ALGORITMO DINÂMICO DE MONITORAMENTO

O algoritmo elaborado é dividido em dois sub-algoritmos para dinamicamente adaptar a coleta de estatísticas às condições da rede. O primeiro sub-algoritmo tem como objetivo atualizar a frequência com que os metadados são inseridos em pacotes da rede e assim possibilitar que sejam detectados variações relevantes do tráfego sem sobrecarregar a aplicação de monitoramento com informações desnecessárias. Já o segundo processo, atualiza valores utilizados no primeiro algoritmo, que foram estaticamente informados pelo programador, para que a solução não fique tão dependente de informações particulares de um fluxo.

Figura 3.1 – Pseudocódigo do algoritmo para controle do período de coleta

Algorithm 1 Algoritmo dinâmico de coleta de estatísticas

1: $time\_interval \leftarrow current\_time - last\_update\_time$
2: **if** $time\_interval >= \gamma$ **then**
3: $\quad diff \leftarrow current\_byte\_amt - previous\_byte\_amt$
4: $\quad$ **if** $diff < -\Delta$ or $diff > \Delta$ **then**
5: $\quad\quad f.\tau \leftarrow T_{min}$
6: $\quad$ **else**
7: $\quad\quad f.\tau \leftarrow min(f.\tau \cdot \alpha,\ T_{max})$
8: $\quad$ **end if**
9: $\quad$ UPDATEVARIABLES($current\_byte\_amt$, $previous\_byte\_amt$, $time\_interval$)
10: $\quad$ UPDATEDELTA(Δ)
11: **end if**
12:
13: $send\_telemtry\_interval \leftarrow current\_time - last\_telemtry\_time$
14: **if** $send\_telemtry\_interval >= f.\tau$ **then**
15: $\quad$ ADDTELEMTRYTOPACKET($stats$)
16: **end if**

Fonte: Autor

A lógica do primeiro sub-algoritmo funciona de maneira que a cada γ milissegundos é calculado o valor de *bytes* trafegados desde a última vez que esse algoritmo executou. Caso esse valor seja menor que $-\Delta$ ou maior que Δ, *i.e*, o trafego variou significativamente, o período de coleta é modificado para o valor T_{min}. Por outro lado, caso o valor esteja entre $-\Delta$ e Δ, *i.e*, a variação de tráfego não foi significativa, o período de coleta é multiplicado por um número α até chegar a um período de coleta máximo T_{max}. Seu funcionamento é apresentado na Figura 3.1.

No segundo sub-algoritmo, o valor da varável Δ do primeiro método é atualizada com base na média das N últimas diferenças de *bytes* calculadas. Mais especificamente, o novo valor de Δ vai ser uma fração dessa média. A Figura 3.2 mostra o pseudocódigo deste algoritmo.

Figura 3.2 – Pseudocódigo do algoritmo de ajuste do Δ

Algorithm 2 Algoritmo que atualiza o Δ do algoritmo 1

1: $sum \leftarrow sum + byte\_count$
2: $N \leftarrow N + 1$
3: **if** $telemtry\_packets\_sent == N$ **then**
4: $\quad mean \leftarrow sum/N$
5: $\quad \Delta \leftarrow mean/X$
6: $\quad sum \leftarrow 0$
7: $\quad telemtry\_packets\_sent \leftarrow 0$
8: **end if**

Fonte: Autor

3.2 IMPLEMENTAÇÃO

Na implementação deste trabalho foi utilizado a linguagem P4 com a arquitetura *v1model*, a qual além de possibilitar a programação dos blocos operacionais para o processamento dos pacotes, também disponibiliza recursos necessários para a criação da solução, como registradores e primitivas de clonagem de pacotes.

A primeira parte da solução é a definição dos *headers* e a etapa de análise dos cabeçalhos. Foram definidos quatro *headers* ou protocolos, três são conhecidos e utilizados globalmente (*Ethernet*, IPv4 e UDP) e o outro é um protocolo desenvolvido especificamente para a coleta de telemetria (*Telemetry*). A ordem dos protocolos no cabeçalho do pacote é a seguinte: *Ethernet*, *Telemetry* (caso exista), IPv4 e UDP. O *header Telemtry* contém campos de identificação do protocolo seguinte e do fluxo monitorado, como também cam-

pos para guardar a quantidade de bytes e o tempo de medição dessa quantidade. Esses detalhes podem ser vistos na Figura 3.3.

Figura 3.3 – Cabeçalho de telemetria

```
header telemetry_t {
        bit<16> next_header_type;
        bit<8> flow_id;
        bit<32> amt_bytes;
        bit<64> time;
}
```

Fonte: Autor

No bloco de análise são extraídos somente os quatro cabeçalhos mencionados previamente e caso o pacote não tenha essa configuração de protocolos ele é rejeitado e não segue para os blocos posteriores. Os *parses* do protocolo Ethernet e do cabeçalho *Telemetry* são detalhado na Figura 3.4.

Figura 3.4 – *Parsers*

```
state parse_ethernet {
    packet.extract(hdr.ethernet);
    transition select(hdr.ethernet.etherType) {
        TYPE_TELEMETRY: parse_telemtry;
        TYPE_IPV4: parse_ipv4;
        default: reject;
    }
}

state parse_telemtry {
    packet.extract(hdr.telemetry);
    transition select(hdr.telemetry.next_header_type){
        TYPE_IPV4: parse_ipv4;
        default: reject;
    }
}
```

Fonte: Autor

Os cabeçalhos, depois de processados, são enviados, primeiramente, para o bloco de ingresso e, subsequentemente, para o de egresso. É nesses dois blocos que ocorrem os principais procedimentos do trabalho. No bloco de ingresso as principais etapas são:

o uso da tabela *match+action* para atualizar os campos do protocolo IPv4, a contagem de pacotes e de *bytes* que passam pelo *switch* e a implementação dos algoritmos da Seção 3.1 e do processo de coleta de telemetria em banda. Já no bloco de egresso é realizado a clonagem do pacote para enviá-lo a um *host* monitor e a remoção do *payload* do pacote clonado.

Antes de executar os blocos de ingresso e egresso é preciso definir as estruturas de dados necessárias para a realização dos procedimentos deles. A Figura 3.5 mostra os registradores usados para salvar dados em comum à todos os pacotes. Na declaração dos registradores pode se perceber que eles são semelhantes a vetores estaticamente alocados, e todos possuem tamanho MAX_PORTS, já que neste trabalho um fluxo é definido pela porta de saída do pacote. Essa escolha do que define um fluxo foi feita por questões de simplicidade, mas poderia ter sido feito um *hash* da tupla-5 para a diferenciação dos fluxos.

Figura 3.5 – Registradores

```
#define MAX_PORTS 10

// Contadores de bytes e pacotes
register<bit<32>>(MAX_PORTS) telemetry_byte_cnt;
register<bit<32>>(MAX_PORTS) pres_byte_cnt;
register<bit<32>>(MAX_PORTS) past_byte_cnt;
register<bit<32>>(MAX_PORTS) packets_cnt;

// Salva o tempo decorrido desde a última atualização do tempo de
   coleta
register<bit<48>>(MAX_PORTS) obs_last_seen;
// Salva o tempo decorrido desdo último envio de telemetria
register<bit<48>>(MAX_PORTS) gather_last_seen;
// Salva o tempo de coleta
register<bit<48>>(MAX_PORTS) gather_window;

// Usados para a atualização do valor do delta
register<bit<32>>(MAX_PORTS) delta;
register<bit<32>>(MAX_PORTS) n_last_values;
register<bit<32>>(MAX_PORTS) count;
```

Fonte: Autor

Além dos registradores, também é preciso definir uma estrutura do tipo *metadata* que serve para repassar as informações coletados para um pacote clonado. Os campos da estrutura são apresentados na Figura 3.6.

Figura 3.6 – Estrutura *metadata*

```
struct metadata{
    bit<1> cloned;
    bit<8> flow_id;
    bit<32> telemetry_amt_bytes;
    bit<64> telemetry_time;
}
```

Fonte: Autor

Com os cabeçalhos extraídos e com as estruturas de dados adicionais declaradas, o processo para a coleta de telemetria do bloco de ingresso pode começar. Primeiramente a tabela de *match+action* é utilizada para fazer a atualização dos valores do protocolo IPv4. A chave para o *match* é o endereço de destino e o algoritmo correspondecia entre o valor do *header* IPv4 e o programado na tabela é o *Longest Prefix Match* (LPM). Em seguida, os registradores utilizados para contabilizar a quantidade de *bytes* e pacotes são incrementados.

Após as operações básicas no pacote, agora são executadas as ações de atualizar o tempo de coleta (*update_telemetry_insertion_time*) e a de salvar a telemetria na estrutura *metadata* (*save_telemetry*). A implementação do algoritmo para atualizar o tempo de coleta é igual ao algoritmo apresentado na Figura 3.1, sendo a única diferença a escrita e leitura dos registadores.

Antes de entrar na função de coleta das estatísticas da rede, o valor do delta é atualizado. O funcionamento completo foi descrito na Figura 3.2. O principal diferencial do pseudocódigo exposto e a implementação é na hora de calcular a média e o novo delta, visto que o P4 não suporta números *float* e a divisão só funciona com valores conhecidos no tempo de compilação. Logo, a única maneira de realizar a divisão em números não constantes é fazer a opreação *right shift*. Ainda assim, existe a limitação de que o *right shift* só funciona com potências de dois. N Figura 3.7 é apresentado uma equação para contornar essa limitação. Por fim, é importante mencionar que devido as restrições do P4 comentadas anteriormente, o valor do $DivisorAjustado$ na Figura 3.7 é informado antes da compilação do programa.

Figura 3.7 – Equação para fazer *right shift* com valores diferente de potências de dois

$$DivisorAjustado = 1/Divisor * 2^{32}$$

$$Resultado = (Dividendo * DivisorAjustado) >> 32$$

Fonte: Autor

A ultima ação do bloco de ingresso a ser realizada é a de salvar as informações da rede na estrutura *metadata*. Semelhantemente a função de atualização do tempo de coleta de telemetria, esta ação também é realizada em intervalos de tempo. Após ter transcorrido o tempo determinado pelo algoritmo da Figura 3.1, é executado o código para salvar as informações referentes ao tráfego da rede (quantidade de *bytes* e tempo de coleta) na estrutura *metadata*. Elas são salvas nessa estrutura intermediaria pois o destino delas é um *host* monitor enquanto que o rumo do pacote em processamento é um *host* usuário, logo o *header* de telemetria não será inserido nesse pacote. A Figura 3.8 mostra a implementação dessa ação.

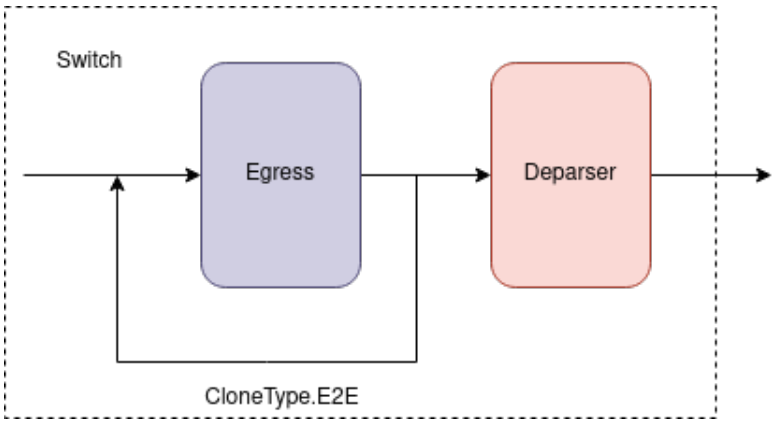
Figura 3.8 – Código para coletar as informações da rede

```
1  void save_telemtry(inout metadata meta, inout standard_metadata_t
       s_m, in bit<32> tel_amt_bytes, in bit<32> amt_packets){
2          bit<48> last_seen;
3          bit<48> window;
4
5          gather_last_seen.read(last_seen, (bit<32>)s_m.egress_spec);
6          gather_window.read(window, (bit<32>)s_m.egress_spec);
7
8          bit<48> now = standard_metadata.ingress_global_timestamp;
9          if(now - last_seen >= window){
10             meta.telemetry_amt_bytes = tel_amt_bytes;
11             meta.telemetry_time = (bit<64>)(now - last_seen);
12             meta.flow_id = (bit<8>)s_m.egress_spec;
13
14             telemetry_byte_cnt.write((bit<32>)s_m.egress_spec, 0);
15             gather_last_seen.write((bit<32>)s_m.egress_spec, now);
16             meta.cloned = 1;
17         }
18     }
```

Fonte: Autor

Terminado o bloco de ingresso, o pacote segue para a etapa de egresso. Nela, se o pacote sendo processado foi marcado na fase de ingresso para ser clonado (linha 17 da Figura 3.8), o procedimento *clone3(CloneType.E2E, ...)* é chamado. Esse método com o parâmetro *CloneType.E2E* faz uma cópia do pacote atual depois que ele passou pelo bloco de egresso. A Figura 3.9 exemplifica o funcionamento dessa função. Após a clonagem, o pacote original sai do bloco de egresso, faz a verificação do *checksum*, adiciona os protocolos modificados no seu cabeçalho e, finalmente, segue pela rede até seu destino.

Figura 3.9 – Diagrama do processo de clonagem E2E



Fonte: Autor

Após o pacote original sair do *switch*, o pacote clonado entra no *pipeline* de processamento direto no bloco de egresso. Ao receber um pacote clonado, o bloco de egresso executa o *match+action* com intuito de redirecionar o pacote para o *host* monitor. Atualizado o destino do pacote, o *header Telemtry* da Figura 3.3 é criado e as informações salvas na estrutura *metadata* durante o processamento de ingresso são adicionados à ele. Além disso, por questões de segurança, os dados clonados e pertencentes ao pacote original são removidos. A Figura 3.10 mostra as ações implementadas no bloco de egresso.

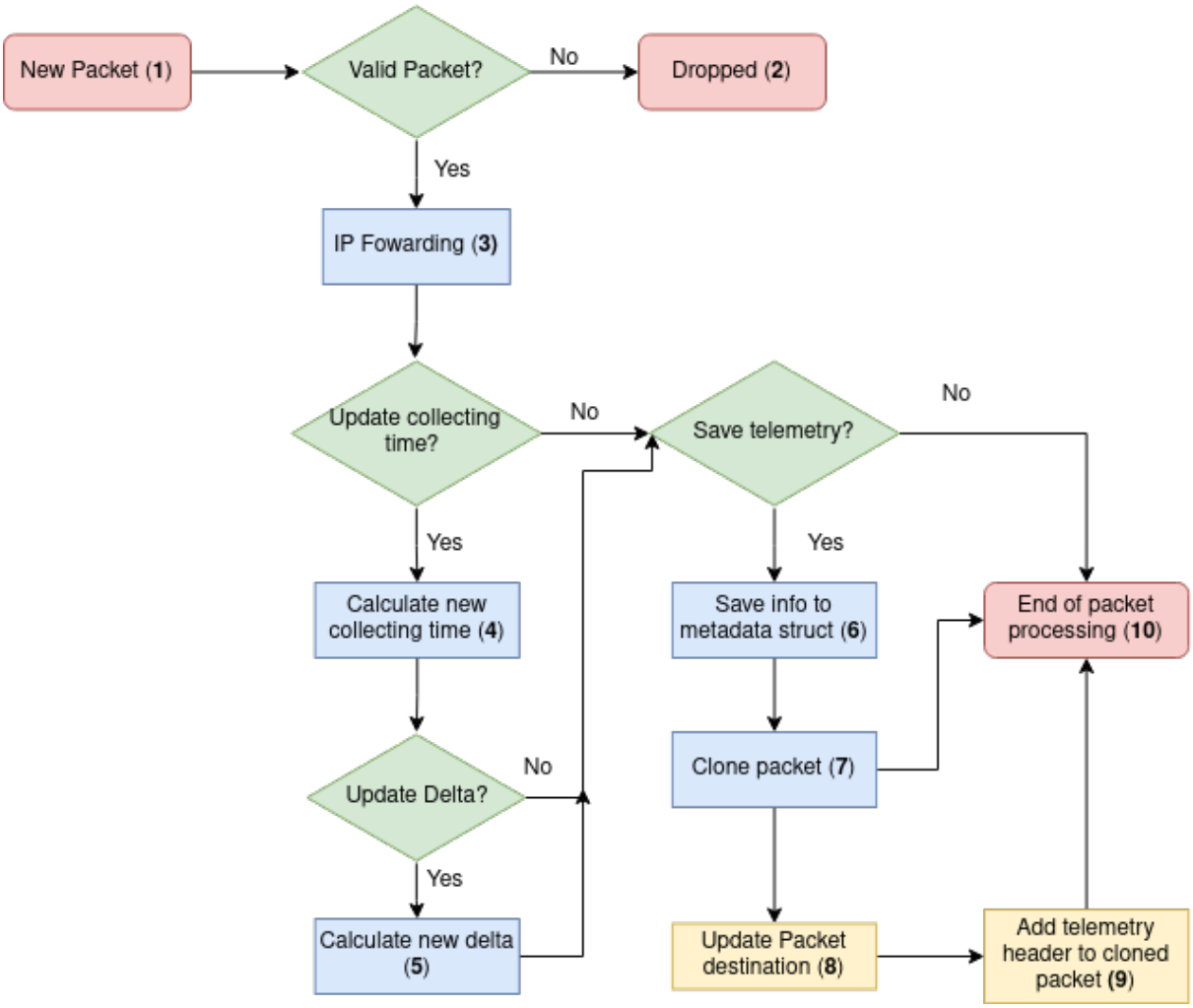
Figura 3.10 – Clonagem e telemetria

```
if(s_m.instance_type == PKT_INSTANCE_TYPE_EGRESS_CLONE){
    clone_table.apply();
    truncate(HEADERS_SIZE); // Remoção do payload
    hdr.ipv4.total_len = 28;
    hdr.udp.len = 8;
    hdr.telemetry.setValid();
    if(hdr.telemetry.isValid()){
        hdr.ethernet.ether_type = TYPE_TELEMETRY;
        hdr.telemetry.next_header_type = TYPE_IPV4;
        hdr.telemetry.amt_bytes = meta.telemetry_amt_bytes;
        hdr.telemetry.time = meta.telemetry_time;
    }
}else if(meta.cloned == 1;){
    clone3(CloneType.E2E, REPORT_MIRROR_SESSION_ID, {meta.
   telemetry_time, meta.telemetry_amt_bytes});
    meta.cloned = 0;
}
```

Fonte: Autor

Com o fim do processamento do pacote clonado e envio dele ao *host* monitor, o método para a coleta dinâmica e eficiente de metadados da rede está finalizado. As etapas possíveis de acontecerem durante o processamento de um pacote são detalhadas no fluxograma da Figura 3.11. Informações adicionais de cada estágio são especificadas na lista posterior a figura.

Figura 3.11 – Fluxograma de um pacote no *switch*



Fonte: Autor

A Figura 3.11 é colorida para facilitar o entendimento geral do processamento de um pacote. Blocos com cor vermelhas servem para indicar o inicio ou fim do processamento dos pacotes. Os verdes são expressões condicionais. Azuis mostram ações realizadas pelo pacote original. E os blocos amarelos são etapas que o pacote clonado realiza.

1) Um novo pacote de dados entra no *switch*.
2) O pacote não possui a configuração desejada, logo ele é descartado.
3) O pacote é considerado válido e segue para ter seu protocolo IP e a porta de saída atualizadas.

4) Foram passados γ segundos desde a ultima atualização, logo é feito novamente a atualização do tempo de coleta com base no tráfego da rede.

5) Como N pacotes já passaram pelo código de atualização do tempo, o novo valor da variável delta é calculado.

6) Transcorreram ω segundos desde o ultimo envio de telemetria, portanto as informações da rede são salvas na estrutura *metadata*.

7) É feito a clonagem do pacote que entrou no *switch* para que um *monitor* tenha acesso às estatísticas da rede ao invés de um *host* usuário. O processamento do pacote original acaba nesse estágio.

8) O destino do pacote clonado é atualizado para o *host* monitor

9) As estatísticas da rede são adicionados ao pacote clonado.

10) Fim do processamento de um pacote.

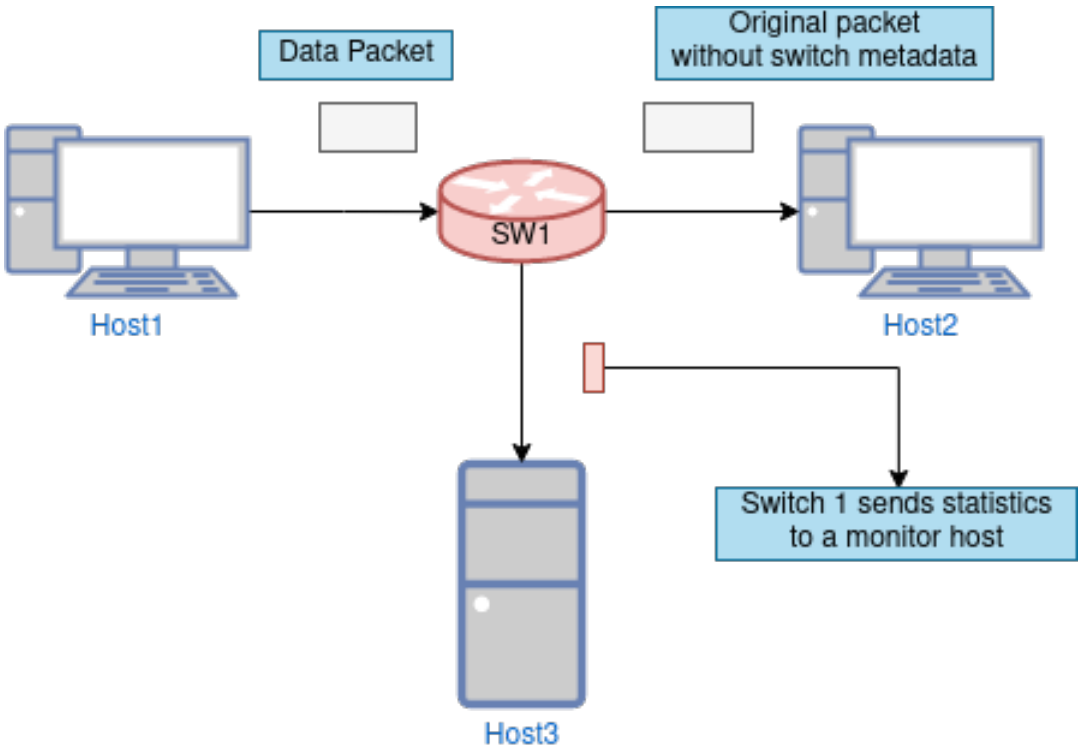
4 AVALIAÇÃO

Neste capitulo serão apresentadas as configurações do ambiente de teste, os experimentos realizados e os resultados obtidos.

4.1 METODOLOGIA

Todos experimentos foram realizados em uma máquina virtual Ubuntu e o ambiente virtual de rede foi emulado pela ferramenta *Mininet*. A topologia de teste criada e a lógica de movimentação dos dados e pacotes é retratada na Figura 4.1. Nessa topologia, o *Host1* se comunica com o *Host2* enquanto que o *Host3* recebe as estatísticas da rede. Todos os *hosts* são interligados por um *switch* P4, onde o código implementado é executado.

Figura 4.1 – Topologia de teste



Fonte: Autor

Para gerar os tráfegos de teste foram usados as ferramentas *iPerf* e a *D-ITG 2.8.1* (AVALLONE et al., 2004). O *iPerf* foi utilizado para inundar a rede com pacotes e assim testar o *throughput* máximo. Já o *D-ITG* foi empregado para testar a acurácia da solução através da criação de diferentes tipos de tráfegos. Em todos os testes, o tráfego era somente UDP. Para capturar os pacotes originais no *host* de destino e os pacote com as

estatísticas no *host* monitor foi utilizado o *software Wireshark*. O propósito dos experimentos realizados é o de demonstrar a eficiência e acurácia da técnica desenvolvida em reportar estatísticas de rede. No caso deste trabalho a estatística reportada foi a vazão de pacotes.

Com intuito de demonstrar que a solução é dinâmica e de que possui boa acurácia, foram gerados três tipos diferentes de tráfegos. O primeiro é um tráfego constante onde existe uma taxa fixa de pacotes por segundos. O segundo é um tráfego em forma de ”piramide" onde existe em um primeiro momento um aumento da utilização da rede de forma constante e subsequente um decréscimo do uso da rede, também de forma constante. O último fluxo é extremamente instável, visto que a quantidade de pacotes muda de forma irregular a cada segundo. Todos os testes mencionados tiveram um tempo de experimentação de 100 segundos.

A fim de validar ainda mais o método desenvolvido, foi realizada uma comparação com um cenário base onde as informações da rede são coletadas estaticamente. O formato do tráfego de comparação é baseado no tráfego proposto no artigo (CHOWDHURY et al., 2014). Para comparar objetivamente os dois métodos, foi utilizado o Root Mean Square Error (RMSE) e o *overhead* de *bytes* causado pelo envio da telemetria. Além disso, em ambos os cenários são experimentados valores diferentes de tempo mínimo de coleta de estatísticas (no método estático não seria o tempo mínimo mas sim o tempo fixo de coleta), visto que esse parâmetro impacta na qualidade dos dados coletados (*i.e.* o T_{min} utilizado no algoritmo da Figura 3.1).

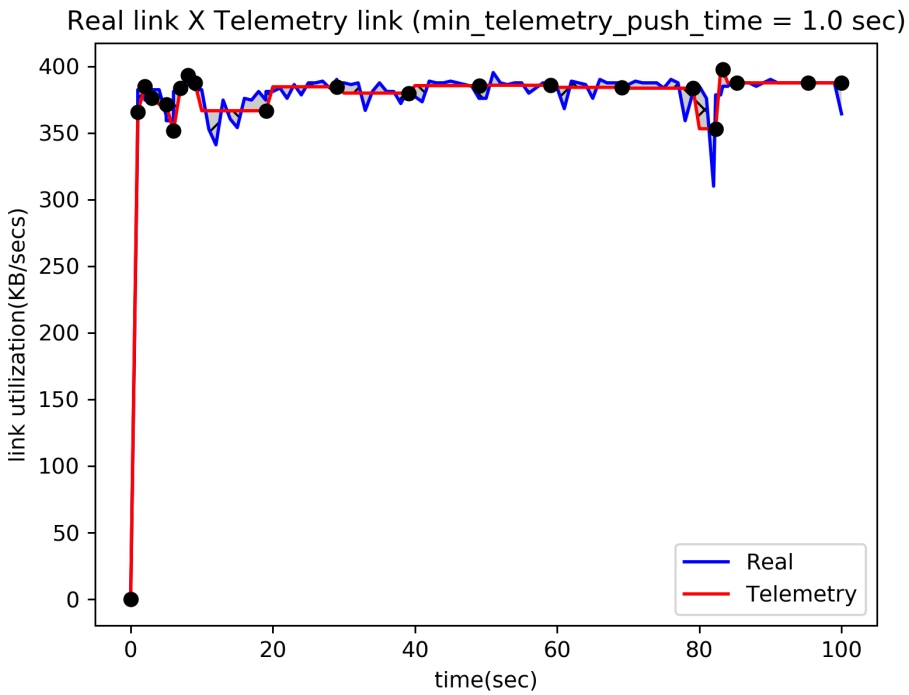
Por fim, para verificar a eficiência da solução, foi computado a vazão máxima e o tempo de processamento médio dos pacotes da técnica proposta. Os resultados obtidos, então, foram comparados com os de um *switch* P4 que somente encaminha os pacotes ao destino deles e um *switch* P4 que faz a coleta das estatísticas de forma estática (o mesmo do teste de comparação mencionado previamente). Todos os resultados são a média de dez experimentos para cada situação.

4.2 RESULTADOS EXPERIMENTAIS

4.2.1 Tráfego variados

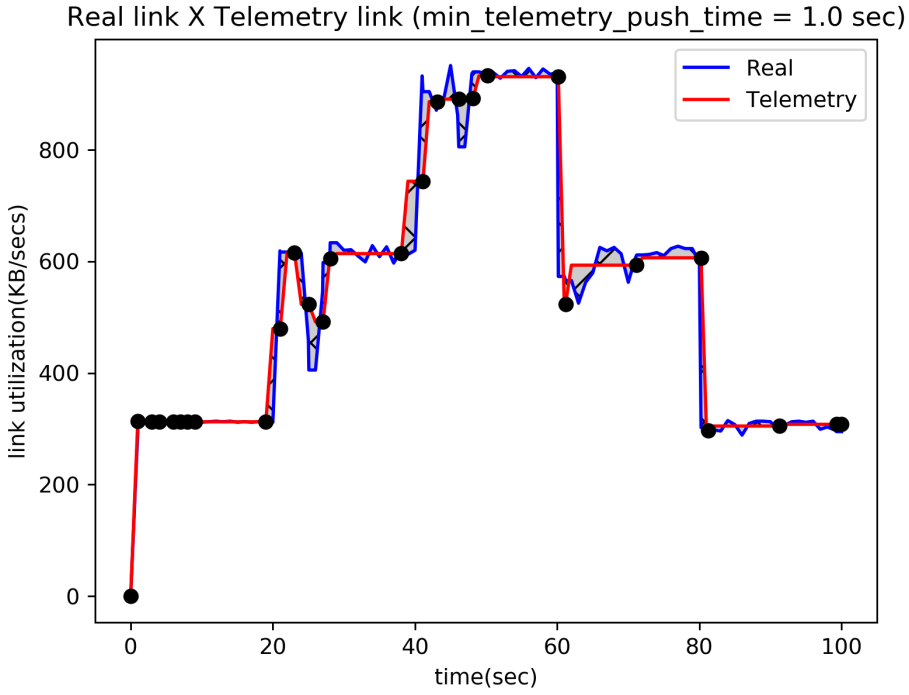
Os resultados dos testes para validar a eficiência e dinamicidade da solução podem ser vistos nos Gráficos 4.1 (constante), 4.2 ("pirâmide") e 4.3 (irregular). Neles são exibidos o tráfego real (linha azul), o tráfego reportado pela telemetria (linha vermelha) e a diferença entre as duas linhas (área hachurada em cinza). Também é detalhado o momento exato em que as telemetrias são enviadas ao *host* monitor (círculos preto).

Gráfico 4.1 – Tráfego constante



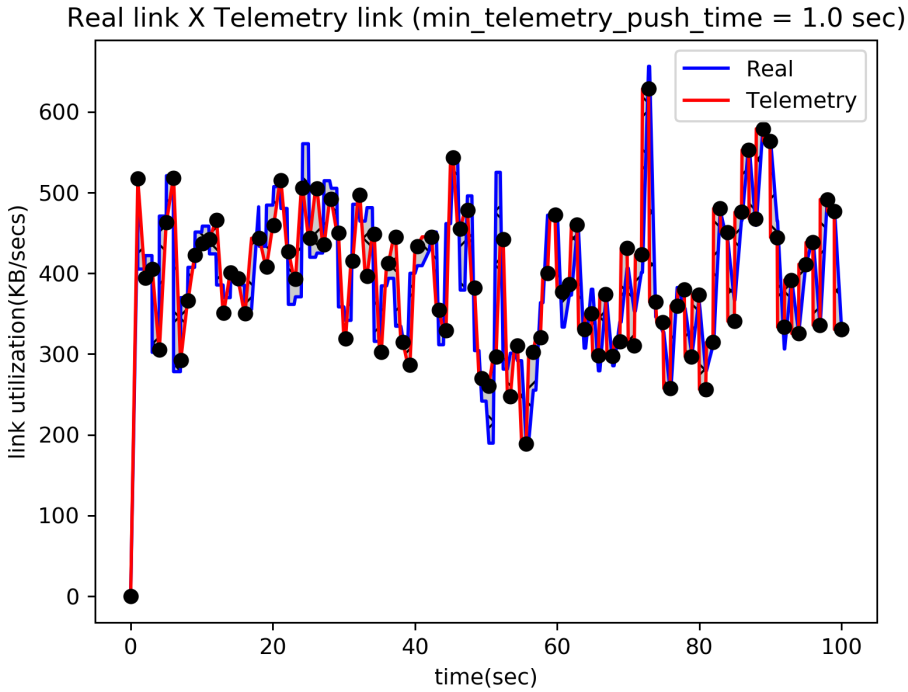
Fonte: Autor

Gráfico 4.2 – Tráfego em forma de ”pirâmide"



Fonte: Autor

Gráfico 4.3 – Tráfego instável



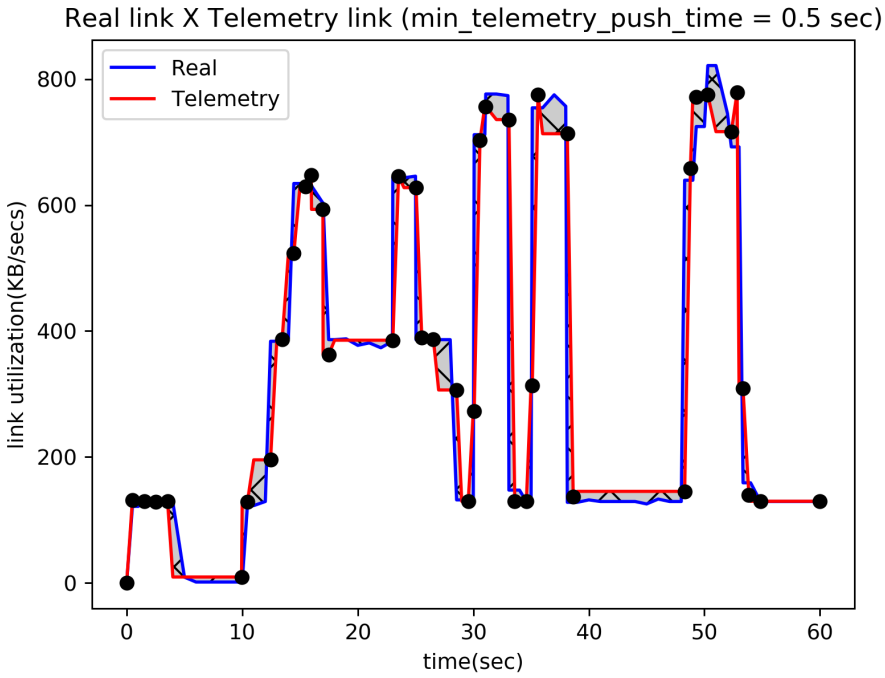
Fonte: Autor

4.2.2 Comparação com outros métodos

Os Gráficos 4.4 e 4.5 mostram os resultados da técnica de coleta dinâmica e estática, respectivamente, onde ambas empregam o tempo de coleta mínimo igual a 0.5 segundos. Nesses dois gráficos, a leitura deles deve ser feita igual à especificada para os gráficos anteriores da seção 4.2.1.

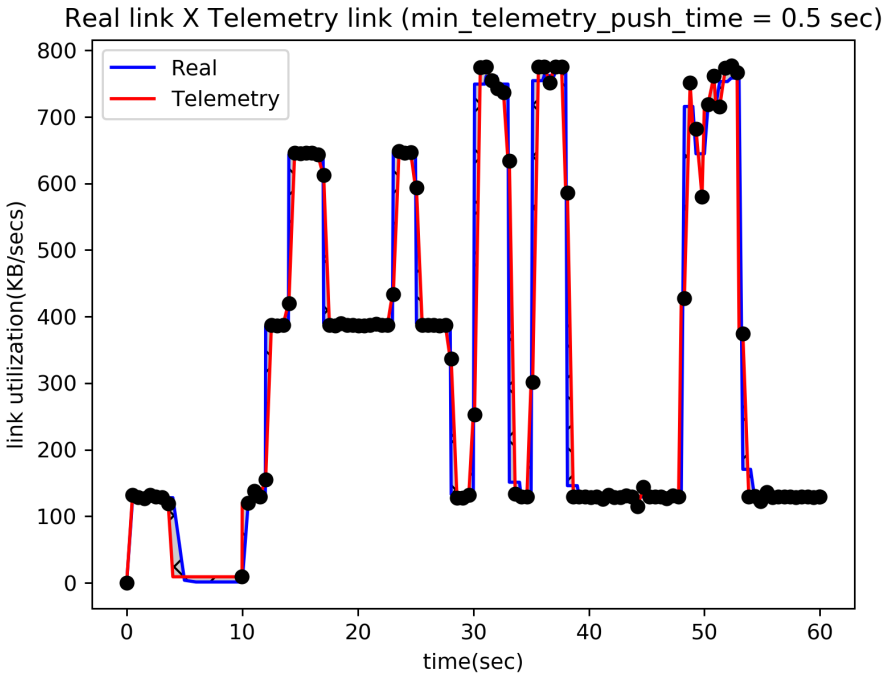
Além dos gráficos que comparam visualmente a diferença entre o valor coletado e o real, foi comparado o RMSE normalizado entre os tráfegos, como também o *overhead* em *bytes* dos cabeçalhos de telemetria. No Gráfico 4.6 pode ser visto esses resultados para a solução proposta. Já no Gráfico 4.7 é apresentado os resultados usando o método de coleta estática. Em ambos os gráficos as barras em laranja indicam o RMSE, enquanto que a barra em verde informa o *overhead* em *bytes* de cada experimento.

Gráfico 4.4 – Monitoramento com o método proposto e tempo mínimo de coleta igual a 0.5s



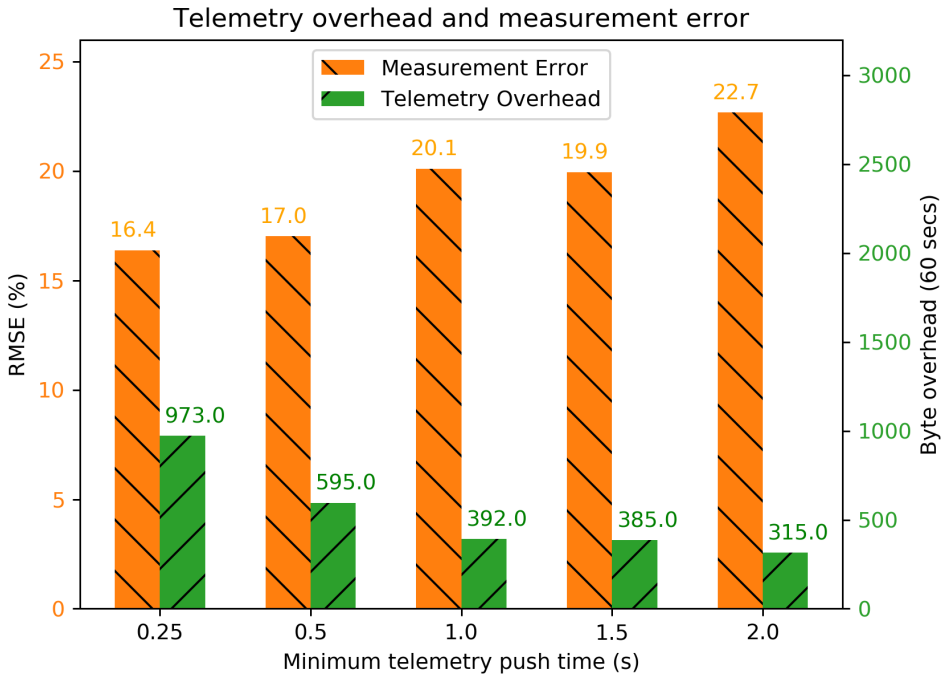
Fonte: Autor

Gráfico 4.5 – Monitoramento com o método de coleta estática e tempo mínimo de coleta igual a 0.5s



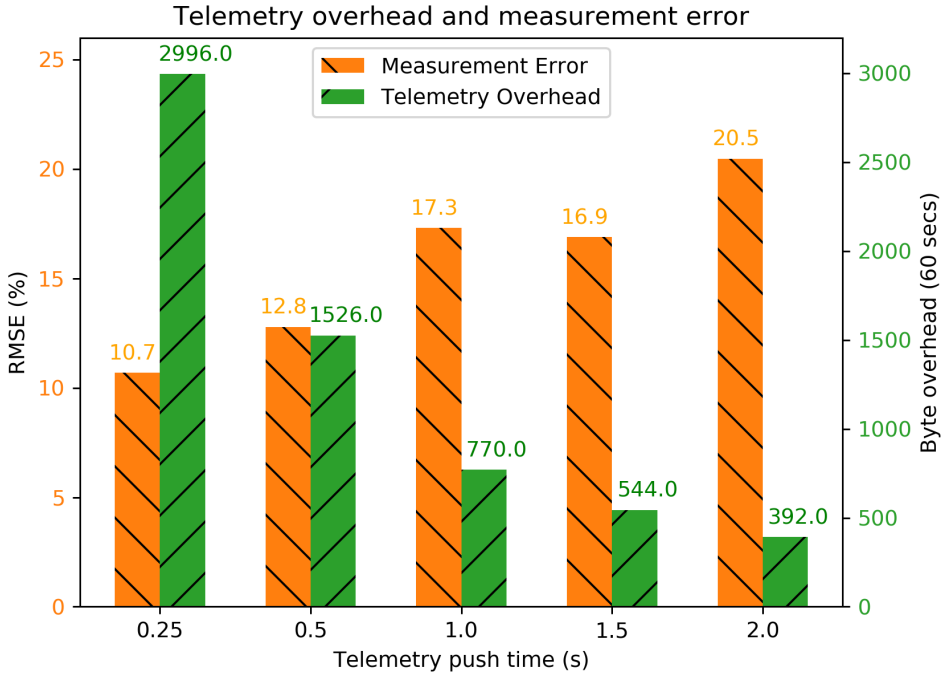
Fonte: Autor

Gráfico 4.6 – RMSE x *Overhead* da solução proposta com diferentes tempos mínimos de coleta



Fonte: Autor

Gráfico 4.7 – RMSE x *Overhead* do método com coleta estática com diferentes tempos mínimos de coleta



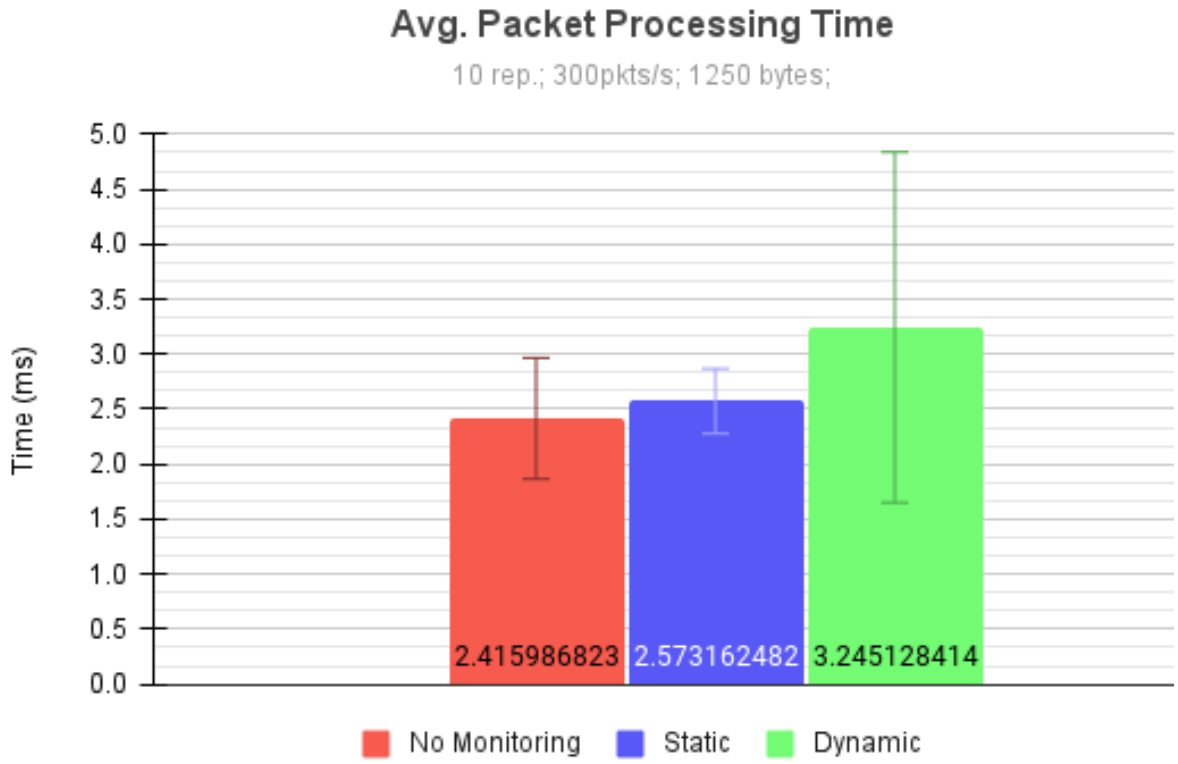
Fonte: Autor

4.2.3 Performance da solução

O último conjunto de testes serve para compreender o quanto a solução implementada afeta o processamento dos pacotes dentro do *switch* P4. Logo, foram realizados testes para verificar o tempo médio de processamento de cada pacote e a vazão máxima da solução. Esses valores foram então comparados com os testes feitos em um *switch* P4 que somente encaminha os pacotes, ou seja não realiza monitoramento, e com outro *switch* que faz a coleta de informações de maneira estática. Para o experimento do tempo médio de processamento, o mesmo tráfego foi usado para cada uma das dez simulações: 300 pacotes por segundos onde cada pacote possui 1250 *bytes* de dados.

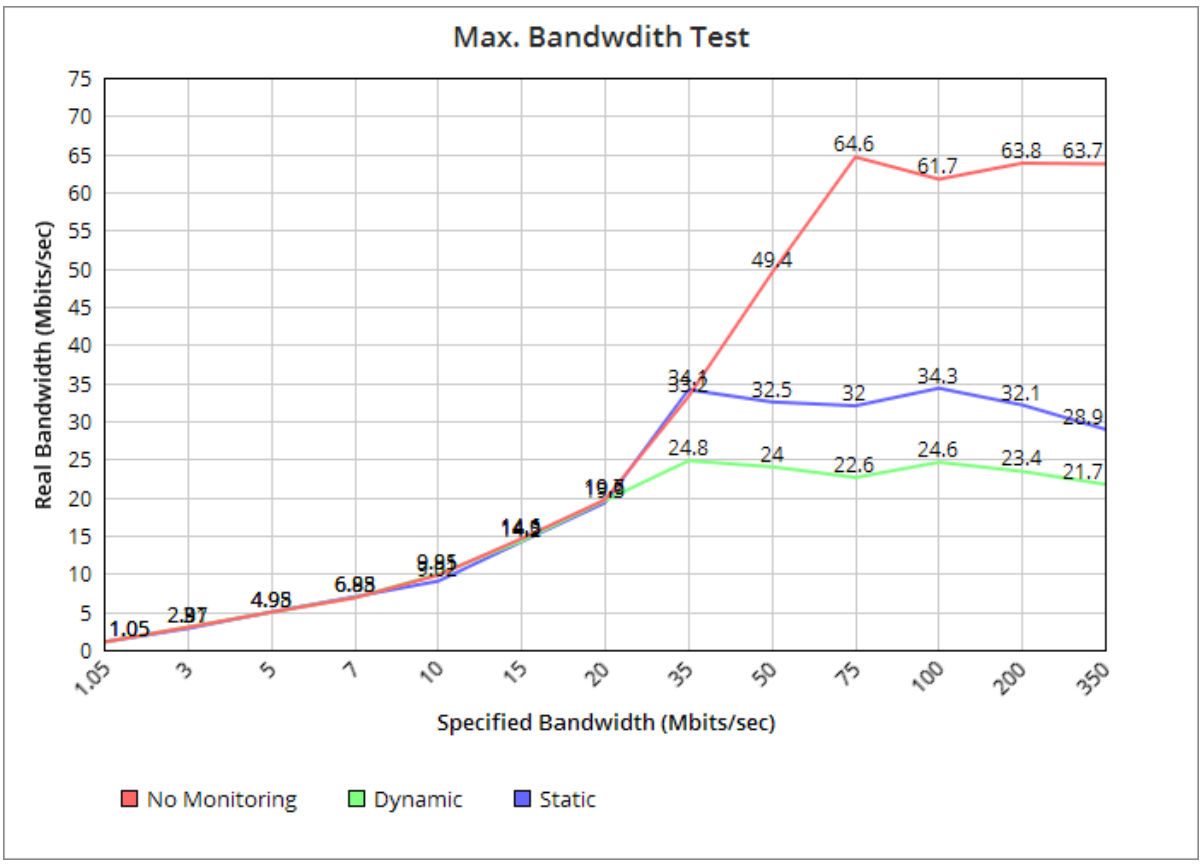
No Gráfico 4.8 pode ser visto a diferença no tempo de processamento de cada implementação e também barras de erros que informam o desvio padrão dos experimentos, enquanto que o Gráfico 4.9 mostra a vazão máxima dos três cenários apresentados.

Gráfico 4.8 – Comparação do tempo de processamento



Fonte: Autor

Gráfico 4.9 – Comparação do tráfego máximo



Fonte: Autor

4.3 DISCUSSÃO

Como pode ser visto nos gráficos da subseção 4.2.1, o método apresentado é capaz de reportar com acurácia o tráfego ao mesmo tempo que ele se adapta para não enviar informações redundantes. Ademais, ele funciona com diferentes tipos de tráfegos, desde o mais estável até o mais volátil.

Na comparação visual entre o tráfego reportado pelo método proposto e o de coleta estática é visível a maior acurácia do cenário com a coleta estática, já que ele não precisa fazer operações para se adaptar. Esse resultado também pode ser visto na comparação entre o RMSE normalizado dos dois métodos. Em contrapartida, o algoritmo proposto possui menor *overhead* de telemetria, o que é esperado, já que para possuir um monitoramento melhor é preciso utilizar mais recursos.

Exemplo dessa alternância de vantagens são os resultados onde o tempo mínimo de coleta é igual a 0.5 segundos. Neles a solução desenvolvida possui RMSE de 17% e usa 595 *bytes* de telemetria enquanto que o método estático tem RMSE igual 12.6% e

overhead de 1526 bytes, cerca de 2.5 vezes mais bytes usados à uma melhora de somente 4.4% na acurácia dos dados coletados em relação à este trabalho. Isso mostra que a solução garante um *tradeoff* bom entre a acurácia das estáticas de monitoramento e o uso equilibrado de bytes para o envio das informações.

Além disso, é possível perceber que em ambos os cenários de teste, a escolha de um tempo mínimo de coleta menor afeta a acurácia da solução positivamente ao mesmo tempo que cria um *overhead* significativo de *bytes*.

Nos experimentos para encontrar o tempo de processamento médio dos pacotes, é fácil visualizar que o *switch* P4 com a solução implementada necessita de mais tempo em média para processar cada pacote, mesmo que em certas simulações o tempo fosse semelhante ao das outras implementações. O motivo do aumento do tempo em relação aos outros cenários, se dá pelo fato de que novas ações foram adicionados ao *switch*, que agora, além de realizar o encaminhamento, precisa fazer diversas operações aritméticas e de escrita e leitura de registradores. Logo, é esperado que existisse um atraso maior ao processar cada pacote.

Semelhantemente, os resultados dos testes de *throughput* mostram que a solução proposta possui uma piora na performance comparada aos outros cenários. Pode ser visto que a vazão máxima da solução é bem menor que o cenário base, a qual somente faz o encaminhamento dos pacotes, e cerca de 10 Mbits/sec menor que a implementação do monitoramento estático. Essa perda de vazão máxima é um consequência direta do aumento do tempo de processamento dos pacotes.

5 CONCLUSÃO

Através da implementação da solução proposta foi possível concluir que a linguagem P4 é capaz de fornecer alto controle sobre o processamento de pacotes e consequentemente é uma tecnologia ideal para o monitoramento eficiente de redes de computadores, apesar das limitações sobre a operação de divisão e a inexistência de números com ponto flutuante. Desta forma, este trabalho contribuiu de maneira significativa para o estado da arte da coleta de estáticas de rede em planos de dados programáveis.

A avaliação da solução apresentou resultados que demostram a sua validade, uma vez que o método é capaz de coletar estatísticas da rede com alta acurácia independentemente do perfil do tráfego e, paralelamente, ele não implica em um atraso elevado no tempo de processamento dos pacotes. Além disso, a solução comparada com o cenário de coleta estática mostrou-se capaz de reportar com acurácia levemente pior à medida que utiliza significativamente menos *bytes* de telemetria.

Ainda que obteve-se sucesso na implementação do algoritmo, seria interessante realizar a comparação dele com trabalhos relacionados e também adaptar o método para ele funcionar com topologias mais complexas e assim possibilitar o monitoramento completo de uma rede e não somente de um *link*. Por fim, o possível acréscimo póstero de recursos à linguagem P4 pode resultar em melhorias na solução atual apresentada, de forma que esta possa vir a ser, inclusive, mais precisa no momento de coletar as informações da rede.

REFERÊNCIAS BIBLIOGRÁFICAS

AVALLONE, S. et al. D-itg distributed internet traffic generator. In: **First International Conference on the Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings.** [S.l.: s.n.], 2004. p. 316–317.

BOSSHART, P. et al. P4: Programming protocol-independent packet processors. **SIGCOMM Comput. Commun. Rev.**, Association for Computing Machinery, New York, NY, USA, v. 44, n. 3, p. 8795, jul 2014. ISSN 0146-4833. Disponível em: <https://doi.org/10.1145/2656877.2656890>.

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.

GOMEZ, C.; SHAMI, A.; WANG, X. **Efficient Network Telemetry based on Traffic Awareness**. TechRxiv, 2021. Disponível em: <https://www.techrxiv.org/articles/preprint/Efficient_Network_Telemetry_based_on_Traffic_Awareness/15057981/1>.

IETF. **Internet Protocol, Version 6 (IPv6) Specification**. 1998. Disponível em: <https://datatracker.ietf.org/doc/html/rfc2460>.

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

MARQUES, J. A. et al. An optimization-based approach for efficient network monitoring using in-band network telemetry. **Journal of Internet Services and Applications**, v. 10, p. 1–20, 2019.

MCKEOWN, J. R. N. **Clarifying the differences between P4 and OpenFlow**. 2016. Disponível em: <https://opennetworking.org/news-and-events/blog/clarifying-the-differences-between-p4-and-openflow/>.

MCKEOWN, N. et al. Openflow: Enabling innovation in campus networks. **Association for Computing Machinery**, New York, NY, USA, v. 38, n. 2, p. 6974, mar 2008. ISSN 0146-4833. Disponível em: <https://doi.org/10.1145/1355734.1355746>.

NAVEEN, S.; KUMAR, R.; KOUNTE, M. R. Ipv6-based network performance metrics using active measurements. In: **Proceedings of International Conference on VLSI, Communication, Advanced Devices, Signals & Systems and Networking (VCASAN-2013)**. [S.l.: s.n.], 2013. v. 258, p. 451–460. ISBN 978-81-322-1523-3.

P4-CONSORTIUM. **P4-16 Language Specification**. 2017. Disponível em: <https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html/>.

____. **INT dataplane specification.** 2020. Disponível em: <https://p4.org/p4-spec/docs/INT_v2_1.pdf>.

SARAI, R. **Software Defined Network and the OpenFlow protocol live (Part 1)**. 2019. Disponível em: <https://medium.com/the-computer-engineer-weekly-code-challenge/software-defined-network-and-the-openflow-protocol-live-part-1-8bbbbb42242e>.

TAN, L. et al. In-band network telemetry: A survey. **Computer Networks**, v. 186, 08 2020.

TANGARI, G. et al. Self-adaptive decentralized monitoring in software-defined networks. **IEEE Transactions on Network and Service Management**, PP, p. 1–1, 10 2018.

TAYLOR, J. T. D. **Network Virtualization: A strategy for de-ossifying the internet**. 2004. Disponível em: <https://www.arl.wustl.edu/netv/>.

YU, C. et al. Flowsense: Monitoring network utilization with zero measurement cost. **Passive and active measurement (PAM)**, v. 7799, p. 31–41, 03 2013.