Pós-Graduação em Ciência da Computação

JOHNATHAN DEZAN VAGO

UMA ESTRATÉGIA PARA ESTABELECER FLUXOS EM REDES SDN-OPENFLOW COM REDUÇÃO DE CARGA NO CONTROLADOR

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE

2017

Johnathan Dezan Vago

Uma estratégia para estabelecer fluxos em redes sdn-openflow com redução de carga no controlador

Este trabalho foi apresentado à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre Profissional em Ciência da Computação.

ORIENTADOR (A): **Prof. Divanilson Rodrigo de Sousa Campelo**

RECIFE

2017

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

V126e Vago, Johnathan Dezan
Uma estratégia para estabelecer fluxos em redes sdn-openflow com redução de carga no controlador / Johnathan Dezan Vago. – 2017.
107 f.:il., fig., tab.

Orientador: Divanilson Rodrigo de Sousa Campelo.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2017.
Inclui referências.

1. Redes de computadores. 2. Redes definidas por software. I. Campelo, Divanilson Rodrigo de Sousa (orientador). II. Título.

004.6 CDD (23. ed.) UFPE- MEI 2017-181

Johnathan Dezan Vago

Uma estratégia para estabelecer fluxos em redes sdn-openflow com redução de carga no controlador.

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre Profissional em 09 de junho de 2017.

Aprovado em: 09 / 06 / 2017.

BANCA EXAMINADORA

_____

Prof. Kelvin Lopes Dias
Centro de Informática / UFPE

_____

Prof. Elmano Ramalho Cavalcanti
Instituto Federal de Pernambuco

_____

Prof. Divanilson Rodrigo de Souza Campelo
Centro de Informática / UFPE
(Orientador)

AGRADECIMENTOS

Agradeço a **DEUS** pela força, sabedoria e oportunidade. Sem **ELE,** nada disso teria sido possível. A **ELE** toda honra e glória para sempre.

A minha amada esposa, **Luiza**, que esteve junto comigo em todas as etapas deste trabalho, sendo minha companheira, sacrificando momentos de lazer e horas de sono para que o objetivo fosse alcançado.

Aos meus pais, **Divino** e **Maria**, por serem exemplo de persistência e por terem forjado meu caráter. À **Mariana**, por ter contribuído na estruturação do texto. Aos **demais familiares** que torceram pelo sucesso deste trabalho, sempre compreendendo as ausências nas reuniões de família.

Ao meu orientador, prof. **Divanilson Rodrigo de Sousa Campelo**, que acreditou na ideia e me aceitou como seu orientando, mesmo com a dificuldade da distância. Ao amigo **Fernando López Rodríguez**, que mesmo morando em outro país sempre esteve acessível, dando apoio diretamente no desenvolvimento do trabalho.

Aos **colegas de mestrado**, em especial aos amigos **Giorge Vanz** e **Marcelo Bastos Roen**, pelo convívio nas semanas que nos deslocávamos para Recife-PE.

Ao Ifes, na figura de seus gestores que apoiaram financeiramente e profissionalmente este projeto. Em especial, ao **Prof Roberto Pereira Santos** e à **Profª Denise Rocco de Sena**, que não mediram esforços para que todas as etapas fossem concluídas.

Aos amigos do **Ifes**, **Giancarlo**, **Renato**, **Anderson** e à **equipe da CTI do Campus Videira do IFC**, que apoiaram a execução deste projeto.

Aos **membros da banca avaliadora** por aceitarem o convite e por contribuírem com seus conhecimentos, enriquecendo ainda mais esse trabalho.

“Portanto, quer comais, quer bebais, ou façais outra cousa qualquer, fazei tudo para a glória de Deus”.

1 Coríntios 10:31

RESUMO

A virtualização de servidores possibilitou que vários serviços fossem disponibilizados a partir de um mesmo equipamento, resultando na diminuição de custos e facilitando a operacionalização de uma infraestrutura de tecnologia de informação. Porém, a infraestrutura de rede, que provê o acesso aos serviços de rede, não evoluiu no mesmo ritmo e passou a ser um ponto crítico na transmissão do tráfego de Internet. Recentemente, o advento das redes definidas por software (Software Defined Networking, SDN) permitiu ao administrador de rede criar, gerenciar e entregar serviços de rede aos clientes a partir de uma interface única. Uma das vantagens proporcionadas em uma rede SDN é a centralização da gerência dos ativos de rede no controlador, o que permite a implementação de políticas a partir de um único ponto. O controlador, entretanto, torna-se um ponto vulnerável e crítico na rede, uma vez que a implementação de tais regras está diretamente ligada a sua disponibilidade. Este trabalho propõe uma estratégia para o estabelecimento de fluxos em uma rede SDN/OpenFlow de modo a diminuir a carga direcionada ao controlador, resultante do envio de consultas feitas por seus clientes. As consultas são essenciais para a aplicação das regras estabelecidas no controlador, que deve estar disponível e responder estas consultas no menor tempo possível. A estratégia proposta neste trabalho foi aplicada em um cenário emulado e em um cenário real em produção criado a partir da infraestrutura de rede de quatro campi de Institutos Federais de Educação no Brasil. Nos dois cenários, o comportamento do controlador foi comparado utilizando clientes OpenFlow tradicionais e clientes OpenFlow modificados com a estratégia proposta. Testes realizados em ambos os cenários constataram que o tempo que o controlador necessita para processar uma mensagem assíncrona packet-in e a carga direcionada ao controlador são menores quando as mensagens foram enviadas utilizando a estratégia proposta neste trabalho.

Palavras-chave: Redes de Computadores. Redes Definidas por Software. *OpenFlow*.

ABSTRACT

The virtualization of servers has permitted that several services being available in the same equipment, which contributed to decrease the cost and allow operationalization of infrastructure of Information Technology. Nonetheless, the network infrastructure, which provides the access to the network services, did not developed at the same rate. Recently, the appearance of the Software Defined Networking (SDN), allowed the network administrator to create, manager and deliver network services to the clients from a single interface. One of the advantages provided in an SDN network is the centralization of the management of network assets in the controller, which allows the implementation of policies from a single point. The controller, however, became critical and vulnerable in the network, since the implementation of those policies is straight connect to its availability. This work purposes a strategy to establish flux in a network SDN/OpenFlow in order to decrease the load sent to the controller that are results from the client's queries. The queries are essentials for the applying of the rules set up in the controller; it must be available and answer those queries in the shortest time possible. The strategy, that this work purposes, was performed in an emulated scenario and real scenario created from the network infrastructure of four campus of Federal Institutes of Education (Instituto Federal de Educação – IFE) in Brazil. In these twos scenarios, the behavior of the controller was compared using traditional OpenFlow clients and modified OpenFlow clients by the proposed strategy. The results from tests performed, in both scenarios, showed that the time needed by the controller to process an asynchronous packet-in message and of the load directed to it, is shorter when the messages were sent using the proposed methodology.

Keywords: Computers network. Software Defined Networking. OpenFlow

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

ACI	Application Centric Infrastructure
ACK	Acknowledgement
API	Application Programming Interface
CPU	Central Processing Unit
DNS	Domain Name System
DoS	Denial of Service
FIN	Finally
FTP	File Transfer Protocol
GUI	Graphical User Interface
HP	Hewlett-Packard
HTTP	Hypertext Transfer Protocol
IAX	Inter Asterisk eXchange
ICMP	Internet Control Message Protocol
ID	Identificador
IETF	Internet Engineering Task Force
IFC	Instituto Federal Catarinense
Ifes	Instituto Federal do Espírito Santo
IP	Internet Protocol – Protocolo de Internet
IPv4	Internet Protocol Version 4
LAN	Local Area Network
LTS	Long Term Support
MAC	Media Access Control
MPLS	Multiprotocol Label Switching

MTU	Maximum Transmission Unit
NAT	Network Address Translation
NETCONF	Network Configuration
NFV	Network Functions Virtualization
NOC	Network Operations Center
OF	OpenFlow
ONF	Open Networking Foundation
OSI	Open System Interconnection
OVS	OpenVSwitch
PCAP	Packet Capture
POP	Points of Presence
POP-ES	Points of Presence in Espírito Santo
POP-SC	Points of Presence in Santa Catarina
QoS	Quality of Service
RAM	Random Access Memory
RDP	Redimensionamento e Deduplicação de Pacotes
REST	Representational State Transfer
RFC	Request for Comments
RNP	Rede Nacional de Pesquisa
RST	Reset
SDN	Software Defined Network
SP	Service Provider
SSH	Secure Shell
TCP	Transport Control Protocol
TI	Tecnologia da Informação

TLS	Transport Layer Security
TOR	Top of Rack
TTL	Time to Live
UDP	User Datagram Protocol
VoIP	Voice over IP
VM	Virtual Machine
VPN	Virtual Private Network
WAN	Wide Area Network

SUMÁRIO

1 INTRODUÇÃO

1.1 Contextualização

Redes de computadores possuem em sua arquitetura equipamentos que proporcionam a interconexão de diferentes locais, com o intuito de que os serviços prestados estejam acessíveis para seus clientes. Cabe ao administrador da rede gerenciar e configurar esses equipamentos de modo que seu comportamento esteja de acordo com a política da organização. Isso é possível a partir da execução de comandos de baixo nível em cada equipamento que compõe a infraestrutura de rede. Administrar redes com um número elevado de equipamentos, geralmente, é uma tarefa complexa de ser realizada, principalmente pelo fato da configuração não ser feita de maneira centralizada.

O paradigma de Redes Definidas por *Software* (*Software-Defined Networking* – SDN) propõe a separação do plano de controle do plano de dados [1]. Essa separação possibilita que a gestão e configuração das regras de encaminhamento dos equipamentos da rede estejam centralizadas no controlador, que tem o conhecimento e acesso a todos os equipamentos, tornando-os meros encaminhadores de pacotes[1].

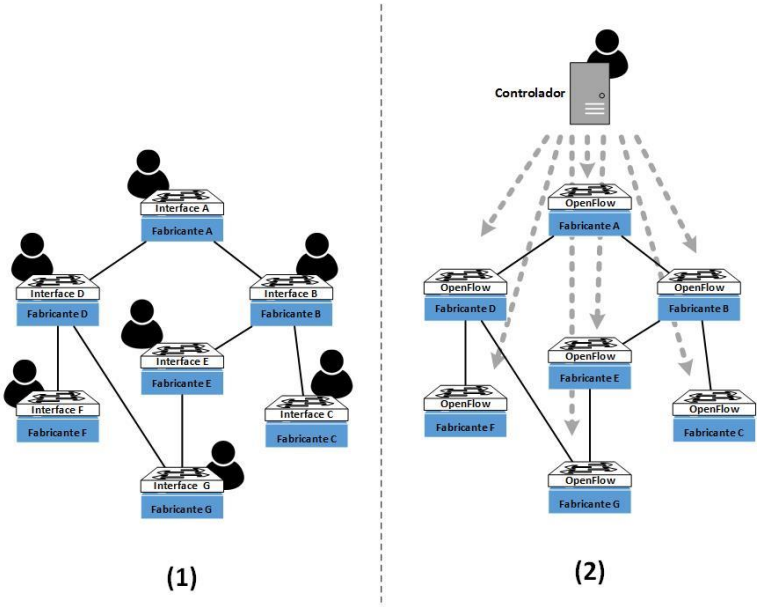
Há diversos tipos de protocolos para o estabelecimento de uma arquitetura SDN [2], entre os quais o protocolo aberto *OpenFlow.* O objetivo inicial do *OpenFlow* é prover para os pesquisadores uma forma de testar novos protocolos em uma rede em produção, sem que seja necessário interferir em seu funcionamento [3]. Desde então, esse protocolo tem sido aprimorado para atender a diversos tipos de ambientes. A *Open Networking Foundation* (ONF) [4] é responsável pela atualização do protocolo e sua documentação [5] e também pelo trabalho junto aos fabricantes de equipamentos de rede para o aumento no número de equipamentos com suporte ao *OpenFlow*.

Semelhante às demais arquiteturas SND, a arquitetura *OpenFlow* é composta pelo controlador *OpenFlow* (que gerencia os equipamentos de rede) e os clientes *OpenFlow* (equipamentos de rede). A comunicação entre o controlador e o cliente *OpenFlow* se dá através da troca de mensagens por meio de um canal de comunicação seguro [6]. O comparativo entre as redes tradicionais e uma rede

[1] O termo pacote é definido pela RFC 1594, como unidade de dados que são enviados em uma rede. Disponível em: <https://tools.ietf.org/html/rfc1594> Acesso em 18 set 2016.

SDN/*OpenFlow* pode ser visualizado na Figura 1.1. Em uma rede tradicional, a interface de gerenciamento é acessada pelo usuário individualmente. Há a possibilidade da gestão[2,3] ser centralizada quando os equipamentos da infraestrutura de rede são do mesmo fabricante e, geralmente, é necessária a aquisição de um produto específico. Já em uma infraestrutura de rede SDN/*OpenFlow*, a gestão é centralizada no controlador e independente do equipamento, desde que ele possua suporte ao protocolo *OpenFlow*. Há diversos controladores SDN/*OpenFlow* gratuitos e com uma vasta documentação disponível na internet.

Figura 1.1 - Comparativo Rede Tradicional (1) x Infraestrutura SDN/*OpenFlow* (2)



Fonte: elaborada pelo autor.

O controlador é responsável pelo estabelecimento da regra de entrada de fluxo, e esta definirá como o equipamento irá tratar um pacote que possua um fluxo correspondente àquela entrada.

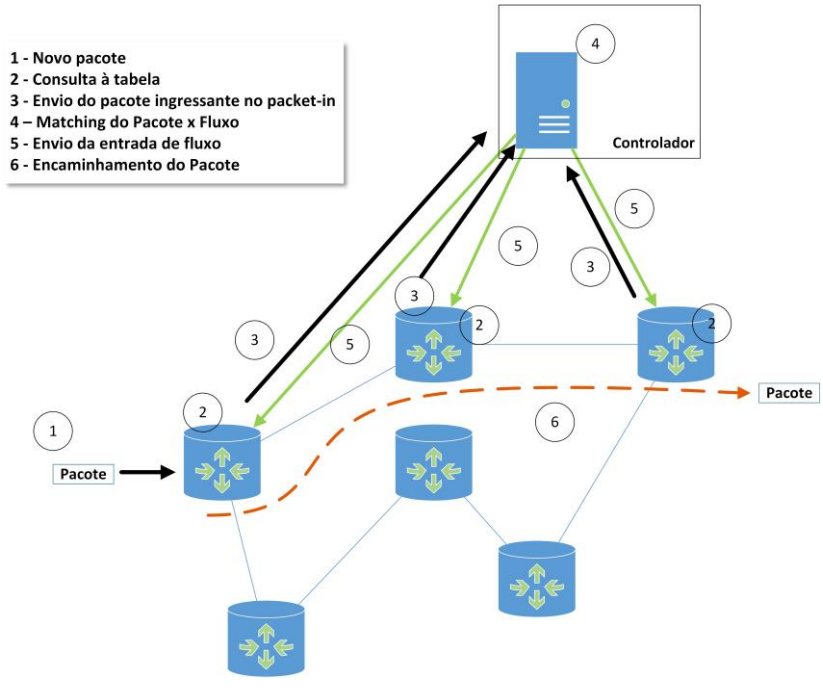
Switches simples possuem uma única regra de tráfego para diferentes tipos de fluxo de dados que ali trafegam [1]. Equipamentos mais robustos possibilitam ao administrador da rede estabelecer regras de fluxo para cada tipo diferente de fluxo de dados. O protocolo *OpenFlow* possibilita o estabelecimento de entradas de fluxo de maneira centralizada por meio do controlador em diversos tipos de equipamentos, simples ou robustos, bastando que ele possua suporte ao protocolo.

[2] Network Assistant – Cisco – Disponível em: < https://goo.gl/P94g5m >. Acessado em março 2016.
[3] Network Node Manager - HP – Disponível em: < https://goo.gl/YDcebg >. Acessado em março 2016.

As regras de encaminhamento inseridas na tabela de fluxo do cliente *OpenFlow* pelo controlador são mantidas por um período de tempo determinado. O administrador pode, portanto, definir quais as regras de encaminhamento e qual o tempo em que elas ficarão armazenadas na tabela de fluxo. Isso proporciona um melhor uso dos recursos de memória e processamento que o equipamento possui.

Na Figura 1.2 estão descritos os passos executados no momento que um pacote ingressa em uma rede *OpenFlow* padrão. Ao ingressar um pacote (1), o cliente *OpenFlow* verifica se há uma regra de encaminhamento na tabela de fluxo que coincida com os campos do cabeçalho do pacote ingressante (2). Caso exista, o pacote é encaminhado pelo fluxo já configurado (6). Caso não exista uma entrada de fluxo para encaminhamento do pacote em seu plano de dados, o cliente *OpenFlow* envia uma consulta ao controlador (3). Ao receber a consulta, o controlador verifica (4) qual o melhor caminho (ou fluxo do plano de dados) a ser utilizado para encaminhamento do pacote e define uma entrada de fluxo específica e a envia (5) ao equipamento que enviou a consulta. Esses passos serão repetidos em todos os clientes *OpenFlow* que fizerem parte do caminho do pacote até seu destino. Feita a inserção na tabela de fluxo de cada cliente, é realizado o encaminhamento (6) do pacote e dos demais que ingressarem enquanto a entrada de fluxo for válida sem a necessidade de nova consulta ao controlador para o fluxo estabelecido.

Figura 1.2 - Funcionamento Padrão do Protocolo *OpenFlow*



Fonte: elaborada pelo autor.

O processo para estabelecer uma entrada de fluxo possui alguns pontos críticos como: o descarte do pacote ingressante por causa do atraso no estabelecimento de um fluxo; o envio de outras consultas ao controlador de um mesmo fluxo de dados que ainda não concluiu o processo descrito na Figura 1.2; e o controle total da rede depender exclusivamente do controlador. Esses pontos podem provocar uma sobrecarga ao controlador e também tornar a rede suscetível à indisponibilidade.

1.2 Motivação

As mensagens enviadas pelo cliente *OpenFlow* ao controlador, referente às solicitações de entradas de fluxos que não existem na tabela do cliente, possuem o pacote ingressante que as motivaram.Nesse pacote estão contidas as informações que identificam o fluxo desse pacote, como o endereço IP origem e destino, por exemplo. Por meio desta e de outras informações, o controlador irá retornar ao cliente *OpenFlow* qual a regra de encaminhamento que deve ser executada. Ou seja, todas as mensagens de solicitação de fluxo possuem o pacote que a originou encapsulado. Além disso, caso outra mensagem seja recebida antes da resposta do controlador, o cliente *OpenFlow* irá enviar nova solicitação de entrada de fluxo.

Nesse caso, ao analisar um cliente recebendo dois pacotes de um mesmo fluxo, já é possível identificar que o envio de duas mensagens seguidas é desnecessário, uma vez que a regra enviada ao primeiro será a mesma a ser aplicada no segundo. Ao analisar uma infraestrutura de rede com vários equipamentos, é possível afirmar que várias mensagens serão enviadas de maneira desnecessária e, sendo assim, o controlador irá processar cada uma dessas mensagens.

É fato que o controlador deve receber as solicitações encaminhadas pelos clientes *OpenFlow* e respondê-las tão logo seja possível. A disponibilidade do controlador é imprescindível para o funcionamento de uma infraestrutura de rede SDN/*OpenFlow*. Porém, as solicitações de fluxo encaminhadas ao controlador podem impactar diretamente em seu desempenho. Por esse motivo, tais mensagens deveriam ser enviadas contendo somente as informações referentes ao fluxo, apenas uma vez.

Este trabalho propõe uma estratégia que diminua o tráfego direcionado ao controlador, reduzindo, assim, a quantidade de mensagens a serem processadas por ele sem que o cliente *OpenFlow* seja privado de enviar as solicitações de fluxo quando essas forem necessárias.

1.3 Objetivos

1.3.1 Objetivo Geral

Propor uma estratégia no envio de solicitação fluxo do cliente *OpenFlow* para o controlador, de modo que seja diminuída a carga no controlador.

1.3.2 Objetivos Específicos

- **i.** Pesquisar o conceito definido pelo protocolo *OpenFlow*;
- **ii.** Modificar a estratégia utilizada para o estabelecimento de novas entradas de fluxo entre o cliente *OpenFlow* e o Controlador;
- **iii.** Definir quais as informações que são enviadas ao Controlador;
- **iv.** Evitar que sejam enviadas simultaneamente novas consultas provenientes do mesmo fluxo de dados ao controlador;
- **v.** Evitar que após o envio da primeira consulta enviada ao controlador novas consultas provenientes do mesmo fluxo de dados sejam enviadas seguidamente e simultaneamente;
- **vi.** Prover ao Ifes uma infraestrutura SDN-OpenFlow como produto deste trabalho;
- **vii.** Verificar o impacto gerado no desempenho dos serviços após a adoção da proposta em cada infraestrutura de rede utilizada.

1.4 Estrutura do Trabalho

A organização deste trabalho está da seguinte forma: no primeiro capítulo, foi apresentada a introdução do trabalho com a finalidade de contextualizar o leitor nos temas abordados por esta dissertação. Ainda nesse capítulo são apresentadas as motivações e justificativas, bem como os objetivos geral e específicos do trabalho.

O segundo capítulo é destinado ao embasamento teórico deste trabalho. Nele são descritas as informações referentes às SDNs, o protocolo *OpenFlow* e a estrutura de suas mensagens utilizadas na comunicação entre o cliente *OpenFlow* e o Controlador. Como também o levantamento da referência bibliográfica que

descreve as soluções existentes, que visam a diminuir a carga do controlador ou melhorar seu desempenho.

A proposta deste trabalho está descrita no capítulo 3. As informações sobre as duas funcionalidades da estratégia proposta são expostas separadamente para que seja possível entender como uma influencia na outra. Além disso, são descritos também os cenários utilizados como ambiente de validação da proposta. Sendo cada ambiente de característica diferente, também foram realizados testes distintos em ambos os cenários.

O comportamento dos cenários, durante a execução dos testes descritos anteriormente, pode ser visualizado no capítulo 4. Nesse capítulo, é realizada a comparação sem e com a proposta desse trabalho para cada cenário. Também são descritos os *scripts* utilizados na coleta desses dados, bem como outras informações relevantes ao cenário.

Por fim, no capítulo 5, estão as conclusões com uma breve análise dos resultados obtidos de cada cenário, a diferenciação da solução proposta e as demais apresentadas, as dificuldades encontradas e os trabalhos futuros.

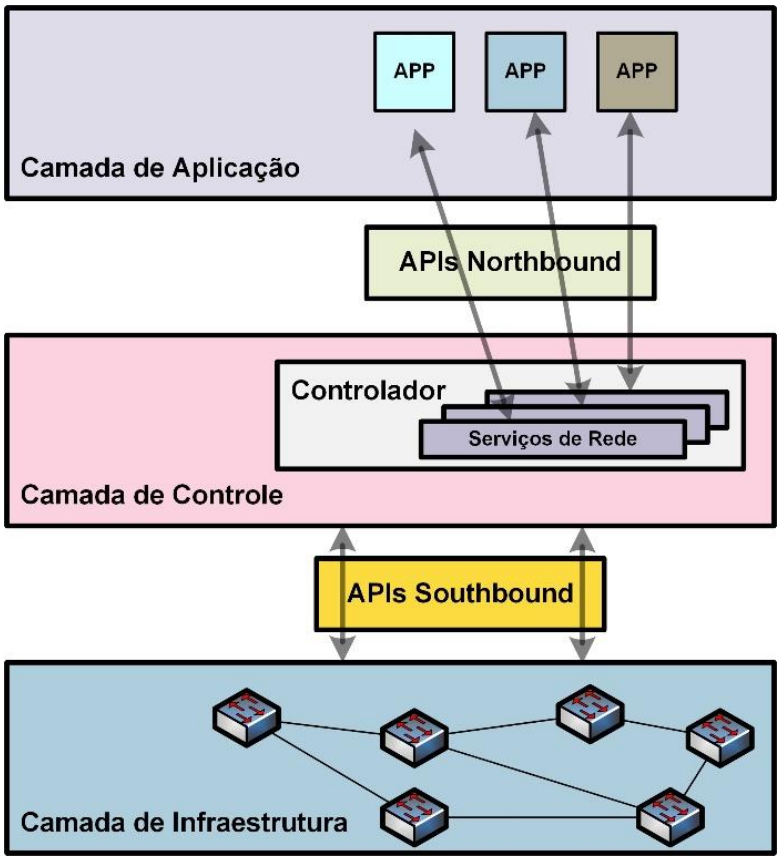
2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo contém o embasamento teórico, em que serão descritas as informações referentes às SDNs, o protocolo *OpenFlow* e a estrutura de suas mensagens utilizadas na comunicação entre o cliente *OpenFlow* e o Controlador. Também serão descritas as soluções existentes, que visam a diminuir a carga do controlador ou melhorar seu desempenho.

2.1 Software-Defined Network (SDN)

SDN tem no desacoplamento do plano de controle do plano de dados a sua principal característica. Ao separar os planos efetivamente, surge a possibilidade de se realizar mudanças em cada um destes de maneira individual. Tais mudanças se dão por meio de APIs (*Application Programming Interfaces*), o que leva a uma definição de que uma rede SDN possui a programabilidade [2] como principal característica. As APIs do plano de controle são denominadas ***API Northbound*** e, por sua vez, uma API do plano de dados é denominada ***API Southbound*** (Figura 2.1).

Figura 2.1 - Arquitetura de uma Rede SDN



Fonte: adaptada pelo autor a partir da estrutura presente em [4].

a) APIs Northbound

A comunicação entre o controlador e os aplicativos e serviços que executam sobre a rede, conforme Figura 2.1, é realizada pela ***API Northbound***. Essa API provê a abstração do funcionamento interno dos dispositivos e da rede em um modo geral para os desenvolvedores. Facilitando ao administrador da rede a integração entre aplicações, como a gestão de configuração de servidores Linux automatizadas e o ambiente onde esses se encontram. Os aplicativos que interagem com essa API podem desempenhar funções na infraestrutura de rede, como *firewalls* e balanceadores de carga.

Devido a esta possibilidade de desenvolvimento diversificado de aplicações, a *Open Networking Foundation* (ONF) [4] tem, por meio da sua comunidade técnica, publicado diversos projetos relacionados à SDN, bem como suas especificações, projetos de código aberto, recomendações técnicas e demais informações necessárias para sua implantação em uma rede de produção.

b) API Southbound

A interface de comunicação entre os equipamentos que compõem a infraestrutura de rede (*switch*es e roteadores) e o controlador são denominadas ***API Southbound***. Por meio dela o controlador pode exercer o controle sobre os dispositivos de rede de modo a implementar a gestão centralizada da rede. Existem padrões de protocolos para proporcionar a comunicação, como o CISCO ACI [7], Contrail [8] e *OpenFlow* [1]; salienta-se, no entanto, que, dentre os padrões citados, o *OpenFlow* é um importante e mais utilizado padrão de interface de comunicação entre as camadas de controle e encaminhamento de uma arquitetura SDN [1], sendo escolhido para este trabalho.

2.2 OpenFlow

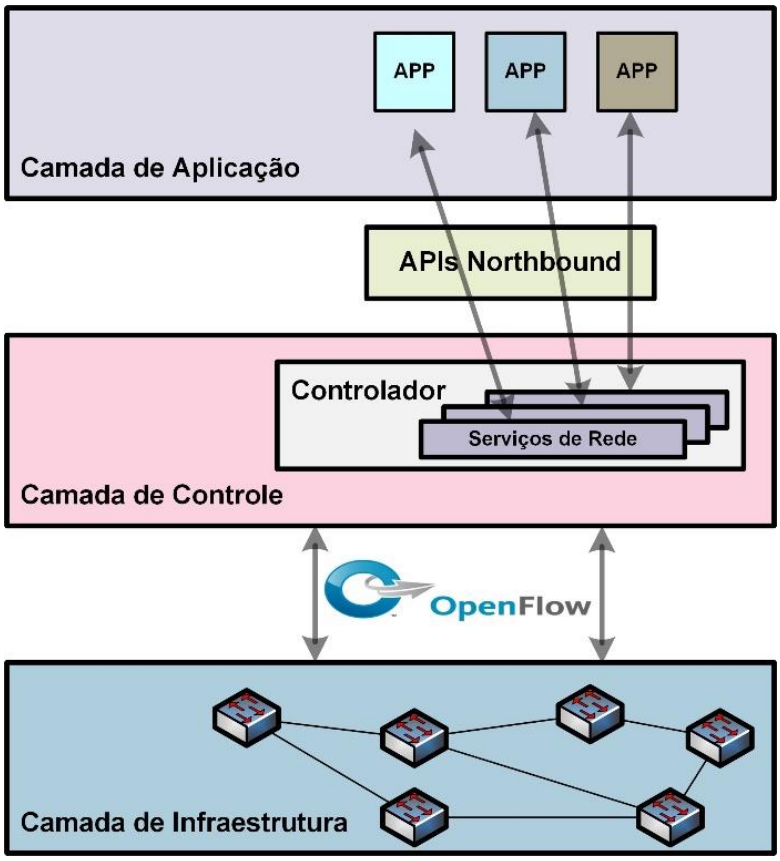
O protocolo *OpenFlow* define a comunicação entre uma API *Southbound* e os equipamentos de rede, criando, assim, uma arquitetura SDN. Esse protocolo é adicionado como uma funcionalidade a mais nos equipamentos de redes. Sua pesquisa se iniciou em *Stanford,* por volta de 2008 e 2009[4] e teve como principal objetivo facilitar a pesquisa e implementação de novos protocolos de rede em uma

[4] Conforme documento “Special Report: OpenFlow and SDN – State of the Union”. Disponível em: https://www.opennetworking.org/images/stories/downloads/sdn-resources/special-reports/Special-Report-OpenFlow-and-SDN-State-of-the-Union-B.pdf Acesso feito em junho de 2017.

infraestrutura existente, conforme [3]. Sua utilização, a partir de então, foi incentivada tornando o *OpenFlow* não apenas uma funcionalidade em um equipamento de rede, mas, sim, em um padrão de implementação de uma rede SDN.

O *OpenFlow* em sua especificação descreve que sua arquitetura (Figura 2.2) é composta por no mínimo um controlador (Plano de Controle) que atua a partir de uma ou mais aplicações (API *Northbound*), de modo a instruir as ações em uma determinada entrada de fluxo. Essas ações definem, por exemplo, qual equipamento e de que forma esse fluxo será estabelecido.

Figura 2.2 - Arquitetura *OpenFlow*



Fonte: adaptada pelo autor a partir da estrutura presente em [4].

Ainda sobre a arquitetura, nota-se que o controlador interage diretamente nos equipamentos via protocolo *OpenFlow* (API *Southbound*). Por fim, uma rede SDN que utiliza o protocolo *OpenFlow* deve, conforme [1], possuir as seguintes características:

i. Ser programável: consiste em habilitar os equipamentos de rede para aplicar em suas tabelas de fluxos as decisões do controlador;

ii. Possuir sua gerência centralizada: os equipamentos de rede são gerenciados, independente do fabricante, a partir do controlador;

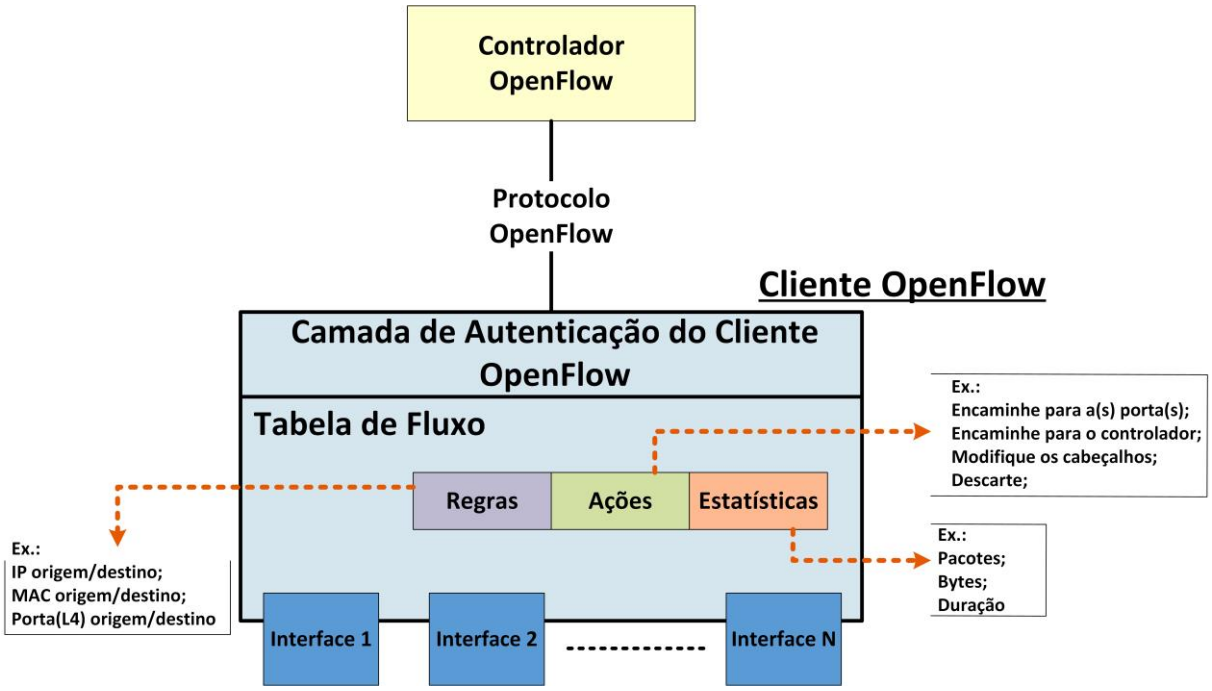
iii. Abstração do plano de dados: o controlador provê essa abstração, pois a partir dele é possível criar aplicações e serviços para gerenciar as entradas de fluxos na rede.

2.2.1 Controlador OpenFlow

O controlador *OpenFlow* define quais comportamentos devem ser adotados nos clientes *OpenFlow* presentes em uma infraestrutura de rede SDN/*OpenFlow*. Sua atuação pode ser comparada a dos sistemas operacionais, uma vez que é ele quem define como a interface de rede do cliente *OpenFlow* deve se comportar.

Como visto no tópico anterior, as APIs são responsáveis pela implementação do cenário SDN. As APIs *NorthBound* interagem com o controlador *OpenFlow*, sendo que, por meio delas, são desenvolvidas as rotinas e definidas as regras de fluxo a serem adotadas na infraestrutura de rede. Essas regras de encaminhamento de pacotes geralmente são definidas no controlador que irá encaminhá-las para os clientes *OpenFlow* (switches) executarem. O controlador possui, portanto, o controle e visualização global de toda rede, o que possibilita automatizar as operações de rede, implantar melhorias e executar modificações de maneira central (Figura 2.3).

Figura 2.3 - Estrutura de um ambiente SDN/*OpenFlow* - Comunicação entre o controlador e os clientes *OpenFlow*. Composição da Tabela de Fluxo do Cliente

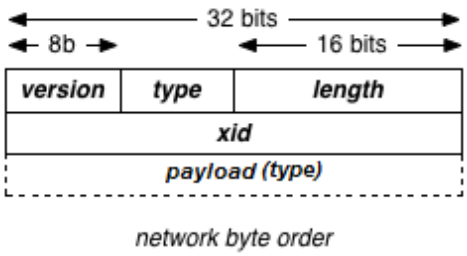


Fonte: adaptada pelo autor a partir da estrutura presente em [1].

2.2.2 Mensagens OpenFlow

No protocolo *OpenFlow*, os clientes comunicam-se com os controladores por meio de mensagens (controller-to-switch, assíncronas e simétrica) [5]. Na estrutura de uma mensagem *OpenFlow* (Figura 2.4), os campos são: (1) "*version*": armazena o valor correspondente à versão do protocolo; (2) "*type*": armazena o valor correspondente ao tipo de mensagem; (3) "*length*": armazena o valor do comprimento da mensagem, a partir do primeiro *byte* do cabeçalho até o seu fim; e (4) "*xid*": armazena o valor correspondente à identificação da mensagem.

Figura 2.4 - Mensagem *OpenFlow* – Estrutura



Fonte: reprodução[5].

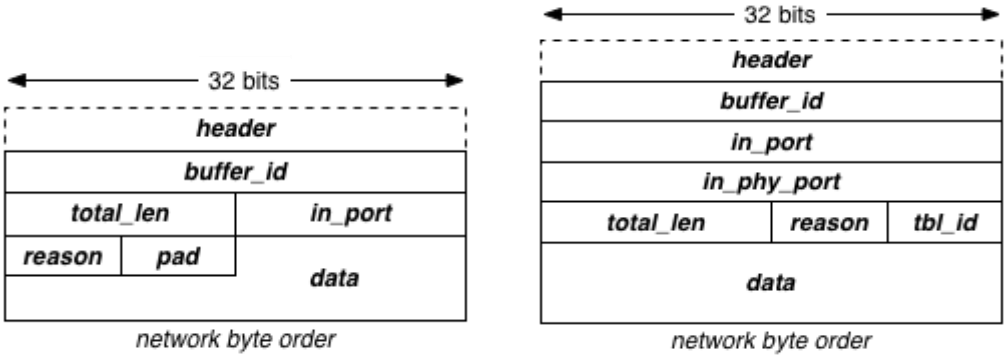
Dentre as mensagens trocadas entre o controlador e o cliente *OpenFlow* as mensagens *packet-in, flow-mod e packet-out* possuem papel importante na

[5] Flowgrammable: "Driving the Next SDN Generation". Disponível em: http://flowgrammable.org/sdn/openflow/message-layer/

implementação de fluxos. As mensagens *packet-in* são enviadas quando não há uma entrada correspondente ao fluxo do pacote ingressante na tabela de fluxo do cliente *OpenFlow* [9]. Ao receber uma mensagem *packet-in*, a resposta do controlador pode ser o envio uma mensagem *flow-mod*, para inserir uma entrada de fluxo na tabela de fluxo do cliente *OpenFlow*, e uma mensagem *packet-out* para esse cliente.

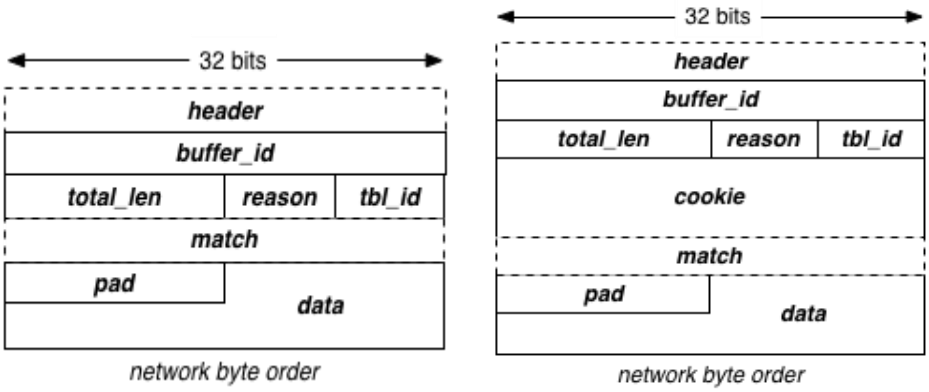
À medida que eventos ocorrem, mensagens são enviadas entre o controlador e o cliente *OpenFlow*; mensagens *packet-in* tem o valor 10 (dez) no campo "*type*", por exemplo. A estrutura das mensagens, a cada nova versão do protocolo, em alguns casos, foi sendo alterada, como a da mensagem *packet-in*, conforme Figura 2.5 e Figura 2.6.

Figura 2.5 - Mensagem *Packet-in OpenFlow* 1.0 e *OpenFlow* 1.1



Fonte: reprodução[4].

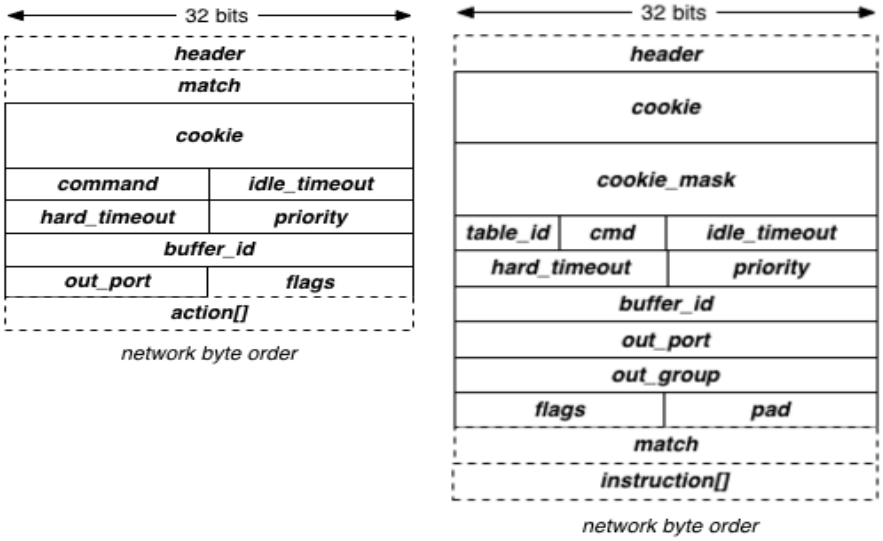
Figura 2.6 - Mensagem *Packet-in OpenFlow* 1.2 e *OpenFlow* 1.3



Fonte: reprodução[4].

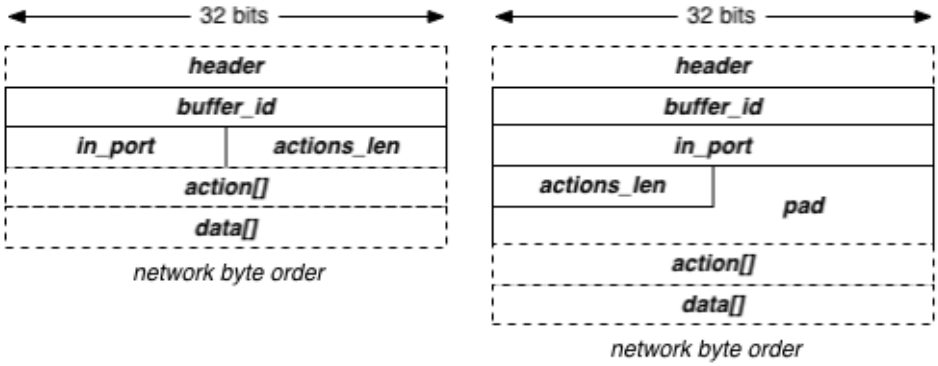
Já a estrutura das mensagens *flow-mod* e *packet-out* (Figura 2.7 e Figura 2.8) é a mesma nas versões 1.1, 1.2 e 1.3, sendo diferente apenas em relação a versão 1.0.

Figura 2.7 - Mensagem *Flow-Mod OpenFlow* 1.0 e *OpenFlow* 1.1[6]



Fonte: reprodução[4].

Figura 2.8 - Mensagem *Packet-out OpenFlow* 1.0 e *OpenFlow* 1.1[7]

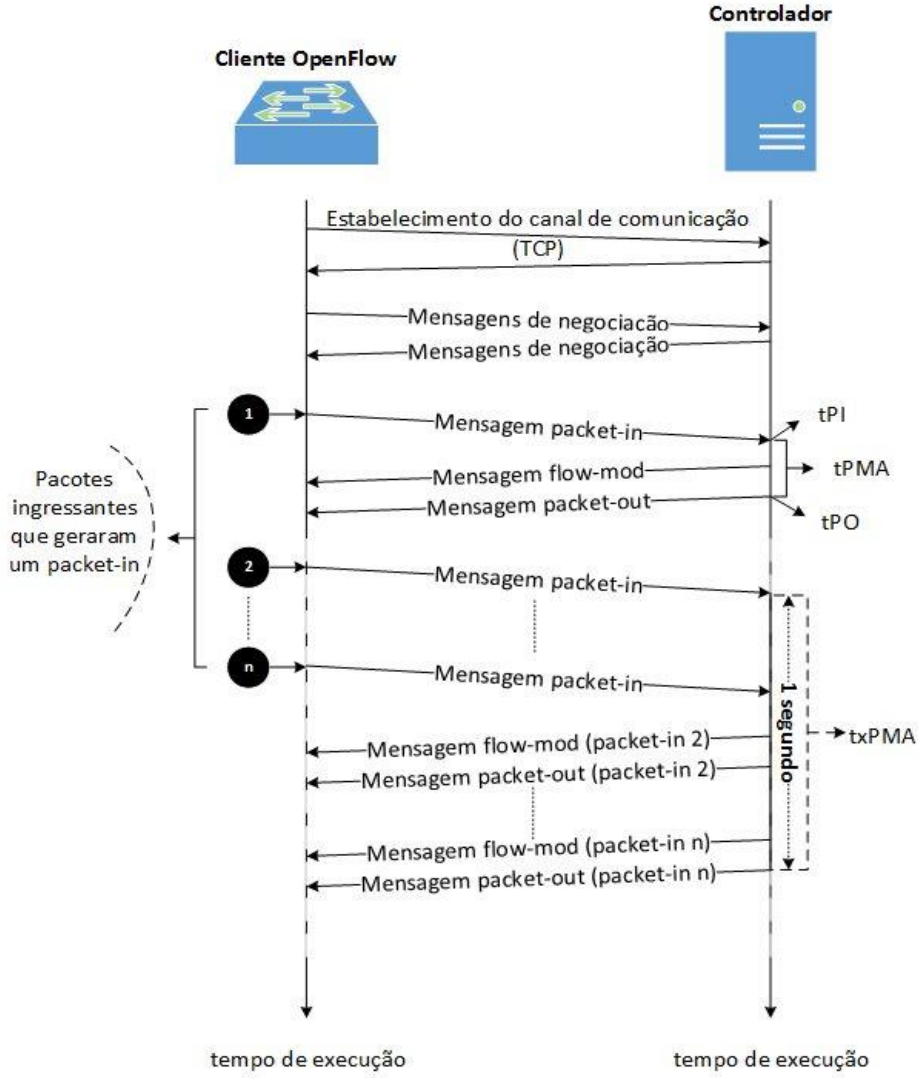


Fonte: reprodução[4].

O *packet-in* é uma mensagem assíncrona e o *packet-out* é uma mensangem *controller-to-switch* [5]. Essas mensagens descritas são enviadas por meio de um canal estabelecido a partir de uma conexão TCP entre o cliente *OpenFlow* e o Controlador. Na Figura 2.9, é possível observar a troca de mensagens entre o controlador e o cliente *OpenFlow*.

[6] A estrutura da mensagem não sofreu alteração nas versões 1.2 e 1.3
[7] A estrutura da mensagem não sofreu alteração nas versões 1.2 e 1.3

Figura 2.9 - Esquema de Comunicação entre o Cliente *OpenFlow* e o Controlador



Fonte: elaborado pelo autor.

Na figura acima, o tPI, tPO e tPMA são, respectivamente, tempo de recebimento do *packet-in*, tempo de envio do *packet-out* e o tempo necessário para processamento de mensagens assíncronas. O txPMA refere-se à taxa de processamento de mensagens assíncronas por segundo.

2.2.2.1 Mensagens packet-in e packet-out

Clientes *OpenFlow* e Controladores trocam entre si diversos tipos de mensagens. Dentre essas, *packet-in* e *packet-out*. O cliente *OpenFlow* envia uma mensagem *packet-in*, a partir da ausência de uma regra correspondente ao fluxo do pacote, na tabela de fluxo, ou pela aplicação de uma regra dessa mesma tabela.

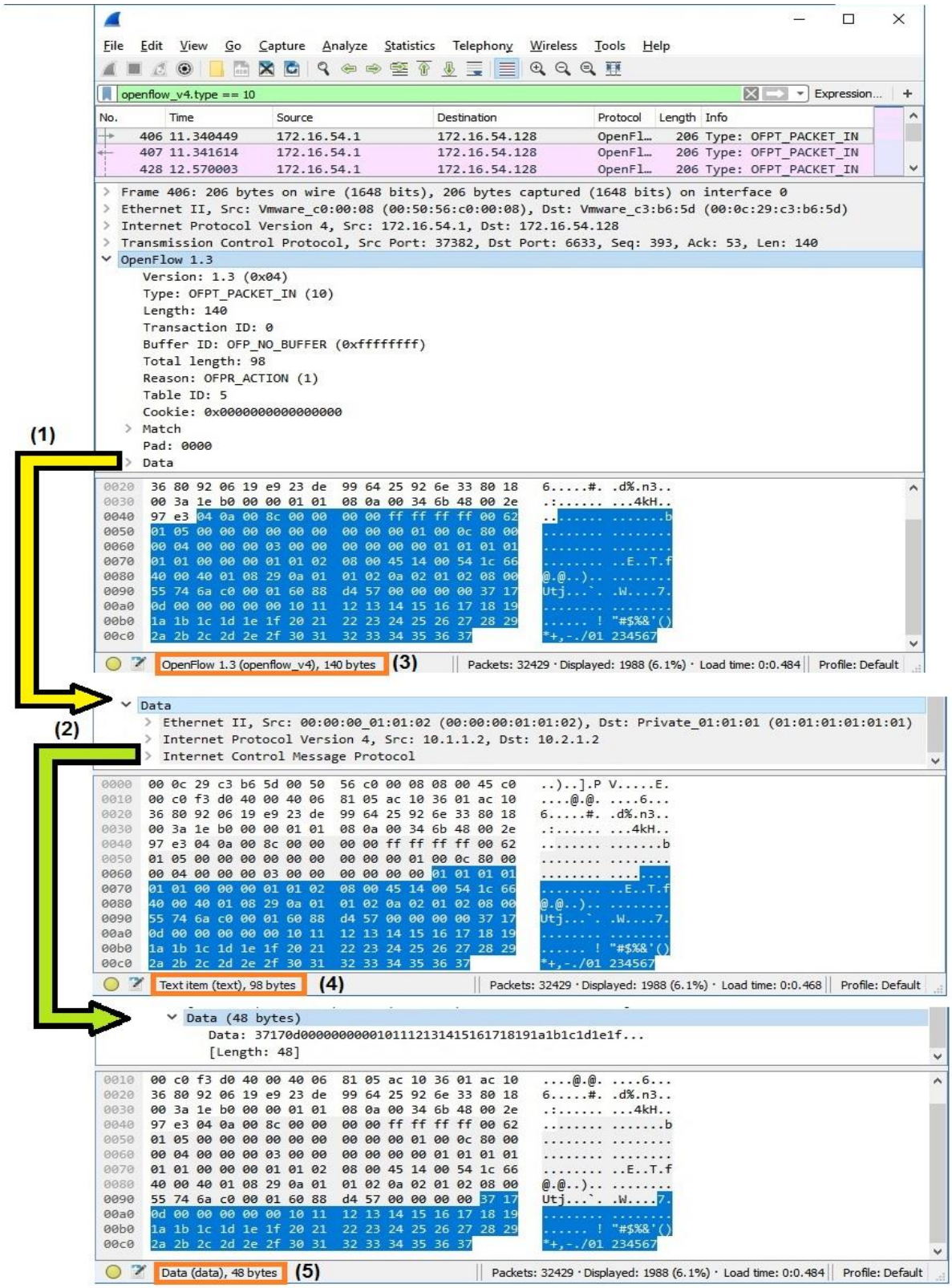
O controlador, por sua vez, ao receber a mensagem *packet-in* do cliente, irá utilizar as informações contidas no pacote encapsulado nessa, para definir qual é a

regra a ser enviada. Ao definir a regra e enviá-la ao cliente, o controlador irá encapsular[8] o mesmo pacote, anteriormente recebido via *packet-in*, na mensagem *packet-out* e enviar ao cliente *OpenFlow.* É preciso salientar que os controladores podem ser configurados para limitar o tamanho da mensagem *packet-out*. Sendo assim, caso o tamanho do pacote a ser encapsulado seja maior que o tamanho definido no controlador, apenas uma parte será encapsulada e o restante será descartado.

Como visto anteriormente, as mensagens *packet-in* e *packet-out* possuem em comum o campo "*data*". Ao capturar essas mensagens, pode-se visualizar melhor cada campo presente na sua estrutura e, ao analisar a captura de uma mensagem *packet-in*, enviado ao controlador, é possível verificar, por meio da ferramenta *Wireshark*, os campos que a compõe (Figura 2.10). Dentre os campos, no campo "*data*" (1), tem-se o pacote responsável por essa consulta ao controlador. A estrutura desse pacote é enviada da maneira como este foi recebido, incluindo seu *payload* (2). Em (3), (4) e (5) estão, respectivamente, o tamanho total do *packet-in* (140 bytes), do pacote encapsulado (98 bytes) e de seu *payload* (48 bytes).

[8] Após a versão 1.3 do protocolo *OpenFlow*, por padrão, se o campo buffer_id possui o valor OFP_NO_BUFFER, o pacote será encapsulado na mensagem.

Figura 2.10 - Captura - Mensagem *Packet-in*

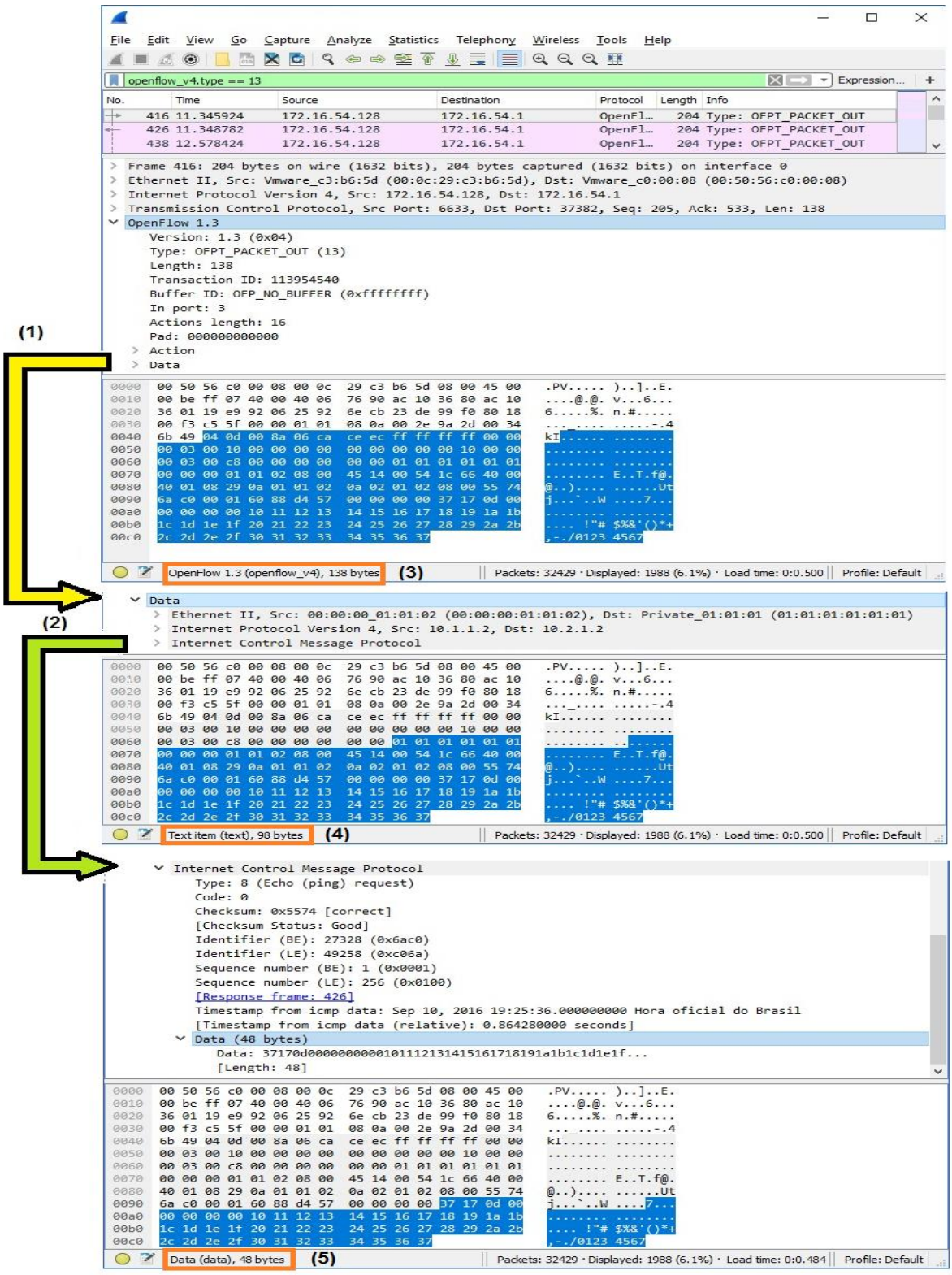


Fonte: elaborado pelo autor.

A mesma observação vale para uma mensagem *packet-out*, pois também é possível verificar os campos que a compõem (Figura 2.11). Ao expandir o campo "*data*" (1), visualiza-se o pacote que originou a consulta *packet-in* e que está sendo

encapsulado, também, no *packet-out*, bem como sua estrutura, incluindo o *payload* (2). Em (3), (4) e (5) estão, respectivamente, o tamanho total do *packet-out* (138 bytes), do pacote encapsulado (98 bytes) e do seu *payload* (48 bytes).

Figura 2.11 - Captura - Mensagem *Packet-out*



Fonte: elaborado pelo autor.

As figuras anteriores fazem referência a uma consulta feita a partir de um pacote ICMP ingressante. Porém, caso o pacote ingressante fosse de outro tipo (TCP, por exemplo), as alterações se dariam apenas no campo “*data*”, visto que ele recebe um pacote de estrutura diferente e, consequentemente, ser de tamanho diferente.

2.3 OpenVSwitch

A virtualização de servidores tornou-se uma prática comum em datacenters. Isso se deve ao fato de uma série de benefícios que esta traz, tais como o rápido provisionamento de recursos, a melhora na disponibilidade em períodos de desastres e a migração para *cloud computing*. Esses benefícios demandam que a rede que interconecta os servidores seja adaptável às mudanças de carga de trabalho e seja reconfigurável. De modo a prover conectividade entre as interfaces virtuais dos servidores, sem que seja necessária a utilização de switches físicos, surgem os primeiros switches virtuais. Esses switches estão inseridos diretamente no *hypervisor*, comportando-se como provedores de conectividade básica da camada de enlace entre as interfaces virtuais, comportamento este semelhante aos switches físicos ToR (*Top Of Rack)* [37].

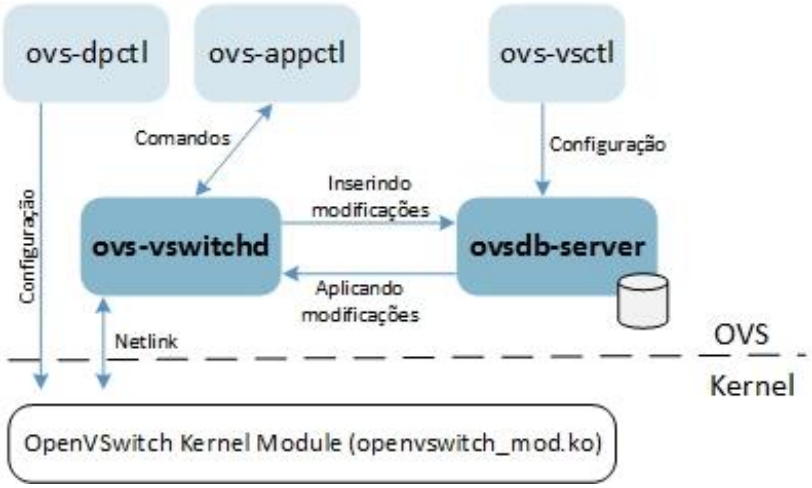
Assim como nos ambientes de virtualização de servidores, a virtualização de rede traz consigo alguns desafios. O compartilhamento de recursos é um deles, uma vez que seus sistemas operacionais não mais contam com *hardware* dedicados, como os switches físicos. As portas físicas geralmente são compartilhadas por mais de um switch virtual. Dessa forma, ter vários switches virtuais desempenhando tarefas diversas, seja provendo a interligação de VMs de diferentes *hypervisors,* ou até mesmo, a interligação dessas VMs com seus respectivos *hypervisors,* faz com que a definição da localização de um switch em uma topologia seja mais complexa.

Contudo, de modo que a virtualização de redes obtenha o mesmo sucesso da virtualização de servidores, é necessário que os switches virtuais não tenham apenas a função de conectar as máquinas virtuais (VMs), mas que possuam funcionalidades semelhantes aos switches físicos.

O *OpenvSwitch* (OVS), que é *Open Source* e multiplataforma, foi desenvolvido com o objetivo de atender à demanda de conectividade proveniente da virtualização de servidores de maneira semelhante a um switch físico. Sua

popularidade advém não apenas de sua plataforma original ser Linux, mas também da limitação dos switches virtuais existentes [24]. O OVS foi projetado e desenvolvido de modo a ser flexível e de uso geral, e não específico para uma determinada tecnologia (como os outros switches virtuais). Os principais componentes do OVS são: *ovs-vswitchd*, *ovsdb-server*, *ovs-dpcl*, *ovs-vsctl* e *ovs-appctl* (Figura 2.12) [9].

Figura 2.12 - Principais Componentes do OVS



Fonte: elaborado pelo autor.

O ***ovs-vswitchd*** é a instância que implementa o switch, sendo responsável pelo encaminhamento de pacotes em conjunto com o módulo do *kernel* do Linux (*openvswitch_mod.ko*) onde está implementado o OVS. O *ovs-vswitchd* consulta suas configurações na base de dados que é gerenciada pelo *ovsdb-server*, um servidor de banco de dados do próprio OVS. O módulo do kernel, utilizado para encaminhar os pacotes em conjunto com o ovs-switchd, pode ser configurado através da ferramenta *ovs-dpcl*. Já o *ovs-vsctl* é responsável pela consulta e atualização da configuração do ovs-switchd. E, por fim, o *ovs-appctl* é utilizado no envio de comandos que serão executados pelas instâncias em execução do OVS. O OVS é composto, também, por outras ferramentas, como por exemplo *ovs-ofctl, ovs-pki* e *ovs-testcontroller*.

Desde sua concepção, o OVS é um *OpenFlow* switch e, por meio do suporte ao *OpenFlow*, ele se torna reprogramável. Para que isso seja realmente uma vantagem, ao analisar o comportamento do OVS em cenários complexos, notou-se que a classificação de pacotes no *pipeline* poderia consumir recursos de maneira

[9] O manual de cada componente está disponível no link: http://openvswitch.org/support/dist-docs/. Acesso feito em junho de 2017.

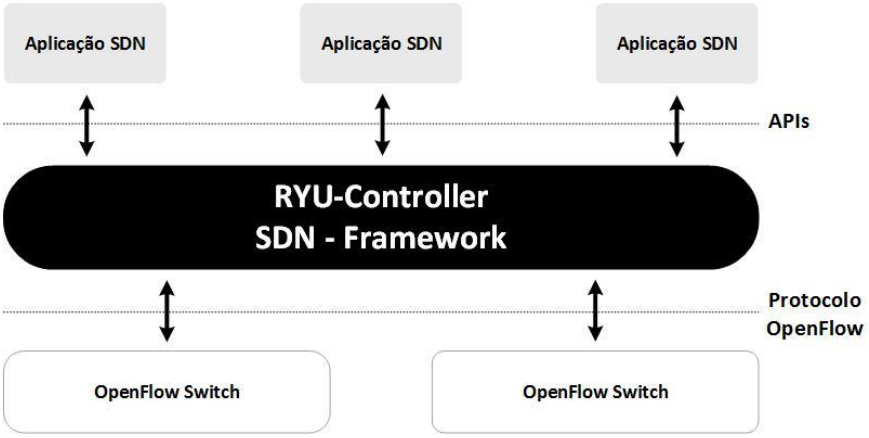
excessiva, principalmente quando relacionadas a outros switches virtuais. Por esse motivo, foi implementado ao OVS o cache de fluxo [24].

2.4 OpenFlow Controller – Ryu Controller

Ryu-Controller é sistema baseado em componentes para SDN. Ele possui suporte a vários tipos de protocolos, dentre eles: *OpenFlow*, NetConf e OF-Config [27]. Baseado em *Python*, o Ryu apesar de possuir algumas desvantagens em relação a outros controladores [25], de um modo geral, seu desempenho é superior aos demais quando são relacionados os valores de desempenho nos quesitos interface, suporte as plataformas, GUI, API REST, rendimento, documentação, modularidade, tempo de mercado, suporte ao *OpenFlow* e ao Quantum [26].

O *Ryu-Controller* possui componentes de software, com APIs bem definidas, para as aplicações SDN a serem utilizadas para o gerenciamento e controle dos ativos de redes. Isso possibilita o desenvolvimento de aplicações que atendam às necessidades de um cenário específico, bem como alterar as aplicações para atender uma demanda proveniente de um novo comportamento da rede. Há também a interface de comunicação com os *switches OpenFlow*, por meio das especificações constantes no protocolo *OpenFlow* [27]. Essa arquitetura pode ser visualizada na Figura 2.13.

Figura 2.13 - Arquitetura do Controlador Ryu



Fonte: elaborado pelo autor.

Em sua instalação padrão, o *Ryu-Controller* possui diversas aplicações de demonstração, utilizando os protocolos suportados. A aplicação *simple_switch13.py*, por exemplo, é uma aplicação que transforma um cliente *OpenFlow* em um mero encaminhador de pacotes utilizando o mapeamento das portas por meio do endereço MAC destas. Outras aplicações estão disponíveis e podem ser utilizadas

como ponto de partida para o desenvolvimento de uma aplicação mais específica para o ambiente.

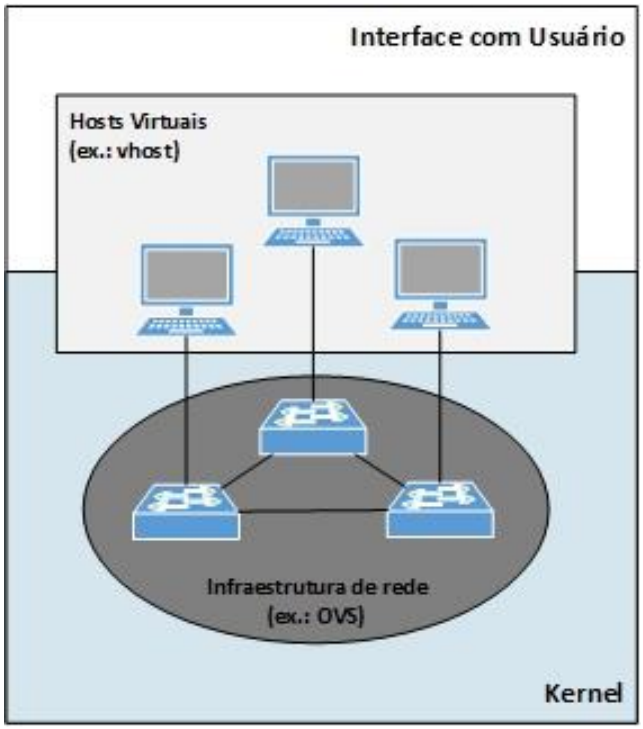
O Ryu-Controller conta com uma equipe de projeto responsável por mantê-lo atualizado, pelo desenvolvimento das aplicações iniciais e também de um livro específico – Ryu-Book – para este controlador [27]. No Ryu-Book, há uma seção chamada "*OpenFlow Protocol*" que descreve como utilizar o ***match***, **instruções** e **ações**, conforme a especificação do *OpenFlow*. As versões do protocolo utilizada foram 1.0 a 1.3. Além dessa equipe, o projeto conta com uma comunidade atuante[10].

2.5 Mininet

Toda proposta de mudança, seja uma nova arquitetura ou um novo protocolo, necessita de um ambiente para avaliação. Nesse caso, um sistema que forneça recursos de virtualização, tais como processos e os protocolos de rede, permitindo que vários nós sejam utilizados, sem consumo excessivo de recursos, tornaria as validações dessas propostas de pesquisas possíveis de serem executadas. Esta é a proposta do *Mininet*: prover um ambiente de rede em uma estação de trabalho simples (ex.: *Notebook*) [28]. O ambiente de rede é composto por links emulados, hosts e switches virtuais. Os hosts virtuais são um conjunto de processos a nível de usuário que recebem um espaço de rede individual. Para prover conectividade entre os hosts, interfaces ethernet virtuais são associadas a esses. Essas interfaces serão interligadas aos switches virtuais por meio de links emulados. A ferramenta *LinuxTrafficControl* é responsável pela configuração da largura de banda desses links. Esse cenário pode ser visto na Figura 2.14.

[10] A Wiki do Ryu-Controller pode ser acessada no endereço eletrônico: http://osrg.github.io/ryu/resources.html

Figura 2.14 - Arquitetura Básica do *Mininet*



Fonte: elaborado pelo autor.

O Mininet é flexível, implementável, interativo, escalável, realista e compartilhável [28]. Com ele, o usuário pode implementar uma nova funcionalidade de rede ou uma nova arquitetura e, ao mesmo tempo, testar os mais diversos tipos de topologia com o tráfego de diferentes aplicações.

Com o *Mininet*, é possível criar, interagir, personalizar e compartilhar uma rede SDN. Ao criar uma rede, o usuário pode definir a quantidade de host, switch e controlador *OpenFlow* que irão compor o ambiente, bem como definir os links que serão criados entre os nós e hosts, com suas respectivas configurações[11].

O *Mininet* tem a seu favor o fato de possui vasto material de auxílio na Internet em relação a outros sistemas que poderiam ser utilizados para realizar as mesmas tarefas, como, por exemplo, o *EstiNet OpenFlow Simulator* [29]. Além disso, o *Mininet* integra o OVS ao ambiente criado por ele.

2.6 Trabalhos Relacionados

A proposta do *OpenFlow* traz benefícios no que diz respeito ao gerenciamento da rede, como a centralização do gerenciamento dos equipamentos

[11] Informação disponível em: <https://github.com/mininet/mininet/wiki/Documentation>. Acesso em: 23 jun 2017.

de rede e a implementação de regras. Contudo, há também uma sobrecarga nessa infraestrutura, principalmente, pela quantidade de mensagens que são enviadas desses equipamentos para o controlador. Nesse contexto, ressalta-se que, por ser uma área com possibilidade de diversas contribuições, algumas delas alinham-se à proposta aqui apresentada.

Em [10], é proposta uma aplicação de controlador modular, multitarefa e interface simples. Por ser modular, a aplicação tem a característica de iniciar ou encerrar aplicações, a partir de uma aplicação em execução. Por ser uma aplicação multitarefa, é possível que, em tempo de execução e sem alterar a aplicação já iniciada, novas aplicações sejam iniciadas (e se comuniquem com outras aplicações) quando a carga enviada ao controlador aumente e sejam encerradas tão logo diminua em tempo de execução. Tais características estão acessíveis, a partir de uma interface de desenvolvimento simples, onde o desenvolvedor pode definir a forma que será realizada a interação com os switches clientes e o controlador. Dito isso, realizados os testes de carga, em comparação aos outros controladores, a aplicação obteve um desempenho considerável e foi capaz de escalar linearmente com núcleos de processamento. Contudo, apesar de conseguir processar uma quantidade considerável de mensagens *packet-in* por segundo, maior que os demais controladores utilizados no trabalho, o controlador fica ainda suscetível ao envio de grandes e repetidas mensagens.

A proposta feita em [11] baseia-se em adicionar dois novos módulos no plano de dados *OpenFlow*: um módulo de migração de conexão e um módulo de gatilho de acionamento. O módulo de migração de conexão identifica quais os locais com conexões TCP irão ou não concluir suas conexões por meio da adoção de cinco etapas: classificar, reportar, migrar e transmitir. Os locais que não irão concluir terão suas conexões migradas antes do envio de qualquer mensagem para o controlador. Já o módulo de acionamento consiste em coletar informações do status da rede e as informações do *payload*, para que, a partir da ocorrência de eventos, as regras de fluxo (ou ações, como *drop* por exemplo) sejam ativadas. Dessa forma, diminui-se o tráfego direcionado ao controlador e evita-se a execução de ataque de “*syn flood*” direcionado ao controlador e demais nós da rede. A proposta afeta apenas pacotes TCP.

Orientando pacotes previamente selecionados por meio de regras armazenadas em switches intermediários, a proposta feita em [12] objetiva ser uma solução escalável e que mantém todo o tráfego no plano de dados. Além disso, para minimizar a sobrecarga no controlador, usa o roteamento de estado do enlace (*link-state*) que permite que os switches aprendam sobre as alterações de topologia sem envolver o controlador. No entanto, essa abordagem pode ter uma carga pesada sobre os switches centrais, que não utilizam um esquema de balanceamento de carga em sua arquitetura.

Diante disso, é proposto em [13] uma mudança do modelo de visão global da rede e do controle do *OpenFlow*, tendo por consequência a diminuição das entradas de tabelas e as mensagens de controle. Nessa proposta, os switches podem tomar decisões baseados em suas regras gerais, inicialmente configuradas e, dessa forma, o controlador só receberá requisições de fluxos específicos.

De maneira similar, em [14] é proposta uma arquitetura de balanceamento de carga que utiliza um algoritmo de particionamento de fluxos que gera regras genéricas a serem instaladas nos *switches* em situações específicas, sem envolver o controlador. Nesse caso, o controlador só será acionado em casos que não estejam previstos no algoritmo implementado.

Semelhantemente, em [15] é proposto um algoritmo de atualização dinâmica das regras juntamente com um esquema de previsão dinâmica, de modo a ampliar o alcance dos fluxos nos quesitos de tempo e espaço. Dessa forma, a quantidade de regras necessárias para o cliente *OpenFlow* pode ser reduzida.

Em [16], é proposta uma arquitetura cujo tempo de resposta e cuja carga direcionada ao controlador são reduzidos, por meio da utilização de fluxos pré-estabelecidos para tráfego sem requisitos QoS (*Quality Of Service*). Dessa forma, apenas os fluxos que possuam requisitos de QoS terão seus pacotes encaminhados para o controlador, para que, assim o controlador crie caminhos ótimos específicos para este tipo de tráfego. Os demais tráfegos são transmitidos sem a necessidade de intervenção do controlador com as regras pré-configuradas.

Já em [17] é abordada a comunicação do cliente *OpenFlow* e o controlador, utilizando controladores passivos, visando à diminuição do atraso no estabelecimento de uma entrada de fluxo nos clientes *OpenFlow* em uma rede com

apenas um controlador. Nesse caso, a diminuição no tempo de resposta ocorrerá devido à inserção de novos controladores e não em implementar mecanismo no tráfego direcionado a estes.

3 ALTERAÇÃO DE ESTRATÉGIA NO ESTABELECIMENTO DE FLUXO OPENFLOW

Neste capítulo serão abordadas as duas funcionalidades da estratégia proposta de forma separada para que seja possível entender como uma influencia na outra. Serão apresentados também os cenários utilizados como ambiente de validação da proposta e os testes utilizados em cada um.

3.1 Mecanismo de Redimensionamento e Deduplicação de Pacotes

O mecanismo de redimensionamento e deduplicação de pacotes (RDP) tem dois objetivos principais: redimensionar o pacote a ser encapsulado no *packet-in*, de modo a enviar apenas as informações necessárias sobre o fluxo que ele possui e que são úteis ao controlador; e evitar o envio de mais de uma mensagem *packet-in* referente a um mesmo fluxo simultaneamente.

3.1.1 Considerações Gerais

No *OpenFlow,* as tabelas de fluxos são responsáveis pelo armazenamento das entradas de fluxo, onde, dentre outras informações, estão as ações de cada fluxo. Isso porque, ao termos um novo pacote ingressante, seu fluxo é, inicialmente, comparado (geralmente, informação origem e destino) nessa tabela. Após isso, serão tomadas as ações que ali estão configuradas.

Uma mensagem *packet-in* é enviada ao controlador quando não há uma entrada na tabela de fluxos do cliente *OpenFlow*, que seja correspondente ao fluxo do pacote ingressante ou quando há uma entrada correspondente a este fluxo na tabela de fluxos que sua ação resulte no envio de mensagem *packet-in* ao controlador. Nesses dois casos, o pacote ingressante será encapsulado na mensagem *packet-in* a ser enviada ao controlador.

As etapas básicas para a composição e envio de uma mensagem *packet-in* são:

i. “**Há entrada de fluxo correspondente?**”: essa etapa realiza a consulta do fluxo do pacote ingressante na tabela de fluxo do cliente. Se sim, será verificada qual(is) a(s) ação(ões) a ser(em) executada(s). Caso negativo, será iniciado o processo de mensagem *packet-in*;

ii. “**A regra tem uma ação de gerar um *packet-in?***”: essa etapa visa a identificar a(s) ação(ões) a ser(em) executadas. Uma regra pode ter mais de uma ação, sendo ela composição de *packet-in*, será iniciado o processo de mensagem *packet-in*;

iii. “**Guardar pacote ingressante no *buffer* interno**”: nessa ação, o pacote ingressante é armazenado no *buffer* do OVS até que seja recebida a resposta do controlador;

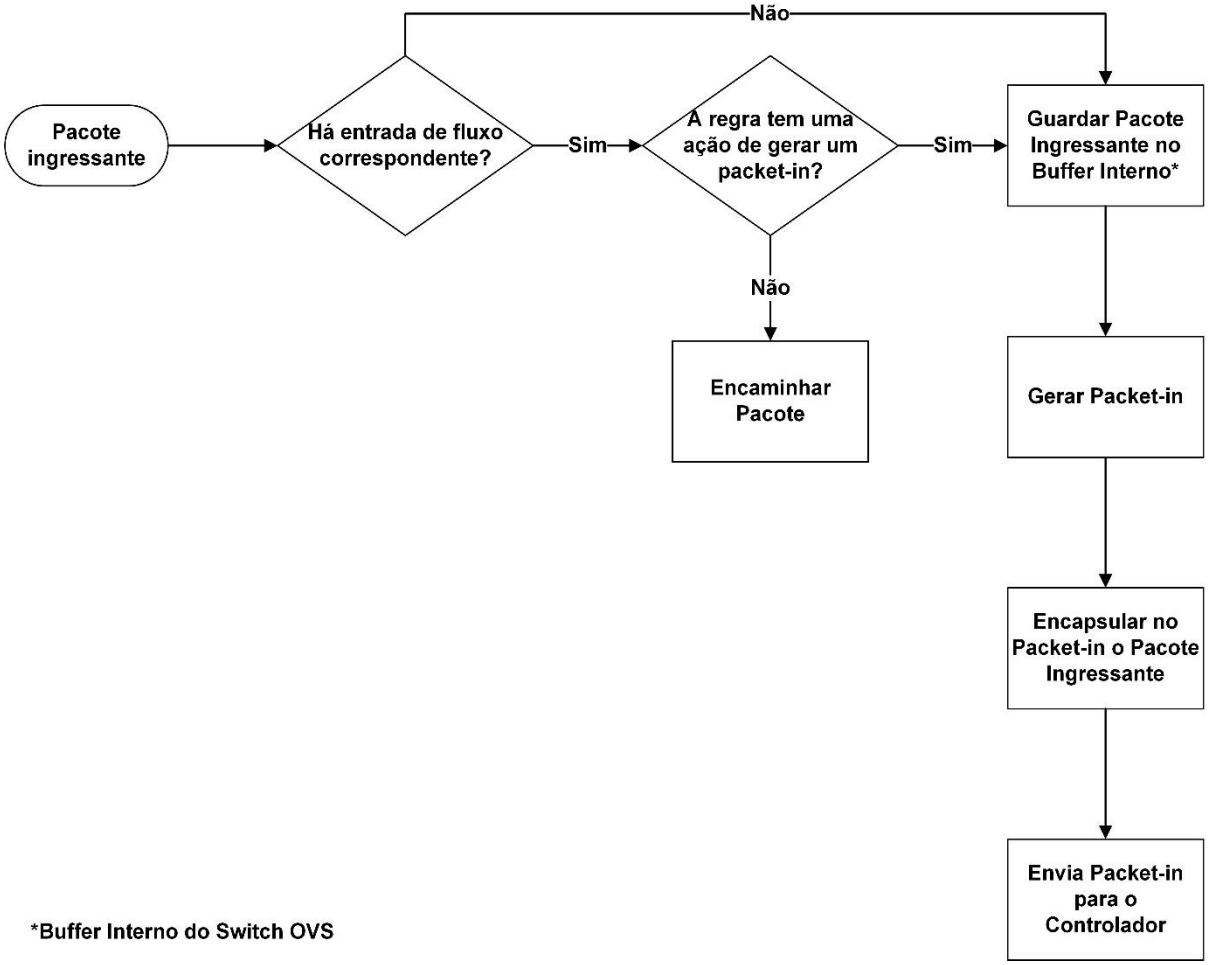
iv. “**Gerar *packet-in***”: nessa etapa, são preenchidos os campos quem compõe uma mensagem *packet-in*;

v. “**Encapsular no packet-in o pacote ingressante**”: no campo “*data*”, é inserido o pacote ingressante. Concluindo, assim, a mensagem *packet-in*;

vi. “**Enviar packet-in para o controlador**”: realizar o envio da mensagem *packet-in* após sua conclusão.

Essas etapas podem ser visualizadas no fluxograma abaixo (Figura 3.1).

Figura 3.1 - Fluxograma do Funcionamento Tradicional - *Packet-in*

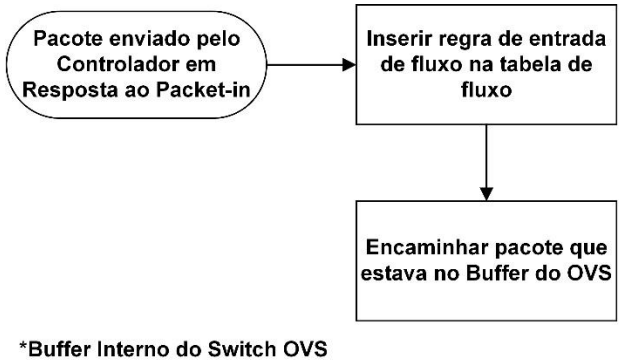


Fonte: elaborado pelo autor.

O controlador, por meio de suas rotinas implementadas, irá enviar ao cliente *OpenFlow* qual a ação a ser adotada para o pacote ingressante encapsulado na mensagem *packet-in* recebida.

Após o envio do *packet-in* ao controlador, o cliente *OpenFlow* manterá o pacote em seu *buffer* até que a resposta do controlador seja recebida. Recebida a resposta do controlador, a ação indicada nesta resposta é aplicada ao pacote que está no *buffer* do cliente *OpenFlow,* conforme Figura 3.2.

Figura 3.2 - Fluxograma da Ação da Resposta do Controlador a um *Packet-In*



Fonte: elaborado pelo autor.

É possível que o controlador tenha enviado a resposta, mas a regra já esteja aplicada. Isso ocorre devido ao fato de vários pacotes ingressantes, de um mesmo fluxo, chegarem simultaneamente ao cliente *OpenFlow,* antes que uma resposta ao *packet-in* enviado ao controlador tenha sido recebida. Também pode ocorrer o envio de várias mensagens *packet-in* referente a um mesmo fluxo, bastando que o pacote ingressante seja maior que o MTU[12] definido no cliente *OpenFlow*. Nesse caso, tal pacote será fragmentado para que seu tamanho seja compatível com o MTU configurado e cada fragmento será enviado ao controlador encapsulado em uma mensagem *packet-in*.

3.1.2 Lógica Geral de Funcionamento

A forma como as mensagens são geradas e enviadas pelos clientes *OpenFlow* impactam diretamente na carga direcionada ao controlador. Como dito anteriormente, o *packet-in* possui em sua estrutura o pacote original ingressante, incluindo toda sua carga útil. E, ainda, pode ocorrer envio duplicado dessas mensagens ao controlador.

[12] MTU (Maximum Transmission Unit) é a unidade máxima de transmissão de um pacote em um segmento de rede. Geralmente, o MTU tem tamanho pré-definido em 1500 bytes.

Uma simples verificação de conectividade entre dois hosts, por meio do envio de pacotes ICMP utilizando a ferramenta *PING,* pode gerar um ou vários envios mensagens *packet-in*, dependendo apenas do tamanho do pacote ICMP a ser enviado.

O tamanho desses pacotes, quer de 64 bytes (tamanho geralmente utilizado), quer de 1500 bytes, pode ser indiferente para o controlador. Assim como em outros tipos de comunicações, como o TCP. O tamanho de uma requisição HTTP de um host cliente a um Web Server, por exemplo, é indiferente para um controlador *OpenFlow*, uma vez que este apenas definirá a regra a ser aplicada para o fluxo presente naquele pacote. Há controladores que implementam regras a partir de dados da camada 4 do TCP/IP e, ainda assim, o *payload* do pacote é irrelevante para o controlador.

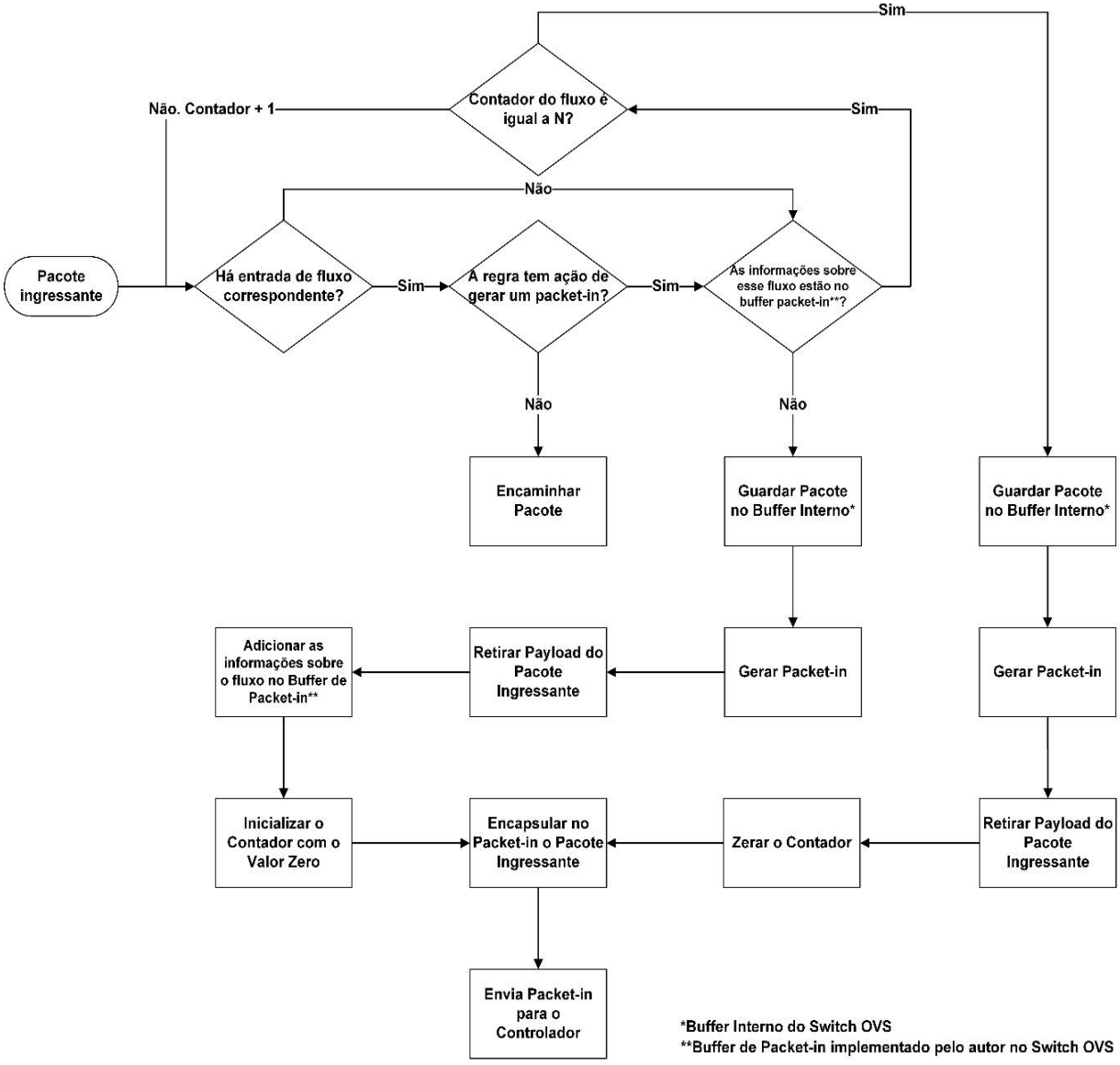
A chegada de vários pacotes referentes a um mesmo fluxo de comunicação, em um curto intervalo de tempo, também pode resultar no envio de várias mensagens *packet-in* ao controlador. Sendo assim, se os pacotes posteriores ao primeiro chegam ao cliente *OpenFlow* antes da resposta à primeira mensagem *packet-in* enviada ao controlador, novas mensagens serão geradas e, posteriormente, enviadas ao controlador. Sendo a mensagem *packet-in* uma mensagem assíncrona, o cliente *OpenFlow* não ficará aguardando o retorno da resposta para um novo envio de *packet-in*. Contudo, é possível que a ausência de uma resposta ao *packet-in*, antes que uma nova mensagem referente ao mesmo fluxo seja enviada, ocorra por outros motivos, quais sejam: atraso no retorno, não recebimento por parte do controlador, perda da mensagem de resposta do controlador, dentre outras indisponibilidades na rede.

No que diz respeito ao envio de mensagens *packet-in* ao controlador, **o tamanho do pacote** a ser enviado, bem como o **intervalo de tempo** de envio, podem ser fatores que influenciem na carga sobre o controlador. A implementação de um mecanismo que defina quais dados do pacote será encapsulado na mensagem *packet-in*, e que restrinja o envio ao controlador de várias mensagens de um mesmo fluxo, em um curto intervalo de tempo, diminuirá a carga enviada ao controlador [18]. E, ainda, tendo em vista que o pacote encapsulado no *packet-in* será encapsulado também na mensagem *packet-out*, como já mencionado, o tráfego proveniente do controlador para o cliente *OpenFlow* também diminuirá a carga sobre

o link utilizado para a comunicação entre eles. Não precisando assim, definir no controlador um tamanho máximo para a mensagem *packet-out*.

A proposta desse trabalho está em: retirar os dados do *payload* da camada 5 (TCP/IP), mantendo o cabeçalho de todas as camadas e implementar um bloqueio de envio de várias mensagens *packet-in* referente a um mesmo fluxo que estejam sendo geradas em um pequeno intervalo de tempo. O fluxograma presente na Figura 3.3 descreve, de maneira geral, o funcionamento do mecanismo proposto.

Figura 3.3 - Fluxograma do Funcionamento do Mecanismo RDP - *Packet-in*



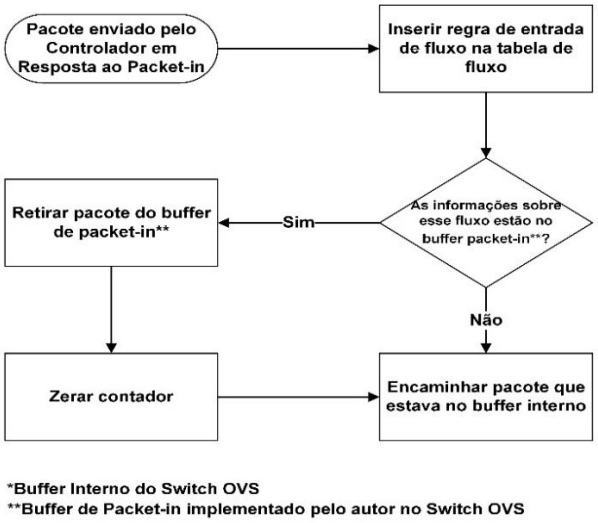
Fonte: elaborado pelo autor.

As etapas inseridas, vistas no fluxograma acima, proveniente da adoção do mecanismo RDP são:

i. “**As informações sobre esse fluxo estão no buffer packet-in?”**: verifica se há no buffer de packet-in uma entrada correspondente ao fluxo do pacote ingressante. Se sim, é verificado quantas vezes já foi feita essa consulta. Caso contrário, é dado prosseguimento no processo de composição de mensagem *packet-in*;
ii. “**Contador é igual a N?”**: etapa onde é feita verificado quantas vezes um fluxo foi consultado e encontrado no buffer de *packet-in*. Caso não seja a enésima consulta, a tabela de fluxo será consultada. Contudo, caso a consulta tenha ocorrido pela enésima vez, o processo para composição do *packet-in* será realizado e posteriormente o valor desse contador será zerado;
iii. **Retirar *payload* do pacote original**”: uma cópia do pacote original (que gerou o *packet-in*) terá sua carga útil retirada antes de ser encapsulado e enviado ao controlador;
iv. “**Adicionar as informações sobre o fluxo no buffer de *packet-in***”: local de armazenamento das informações que identificam o fluxo do pacote original e que foi enviado, encapsulado, no *packet-in*;
v. “**Zerar o contador**”: ação que insere o valor zero para o contador do fluxo inserido ou encontrado no buffer *packet-in*.

Considerando que o *packet-in* foi enviado e recebido pelo controlador com sucesso, e que o Controlador gera o *packet-out* correspondente como resposta, quando o cliente a recebe, ele utilizará o pacote ali encapsulado para extrair as informações de fluxo para consultar o buffer de *packet-in* e, caso exista uma entrada correspondente, ele será retirado desse buffer e o contador será zerado. O fluxograma de funcionamento dessa rotina pode ser visualizado na Figura 3.4.

Figura 3.4 - Fluxograma do Mecanismo RDP - *Packet-out*



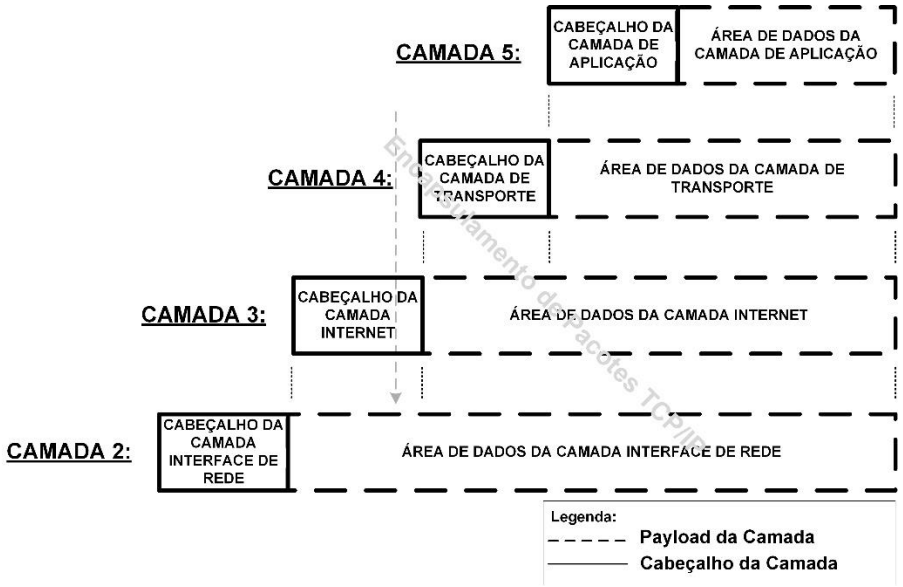
Fonte: elaborado pelo autor.

Porém, nos próximos tópicos, cada abordagem será descrita de maneira mais detalhada.

3.1.3 Lógica de Redimensionamento do Pacote

Assim como a o Modelo de Referência OSI [19], a arquitetura TCP/IP [20] separa a estrutura de comunicação em camadas. O TCP/IP define a quantidade de cinco camadas e cada uma possui características e funções diferentes (semelhante ao Modelo OSI). Uma requisição HTTP, por exemplo, utiliza em sua estrutura as cinco camadas. Há serviços que utilizam pacotes compostos por apenas três camadas. A Figura 3.5 ilustra como as camadas estão relacionadas.

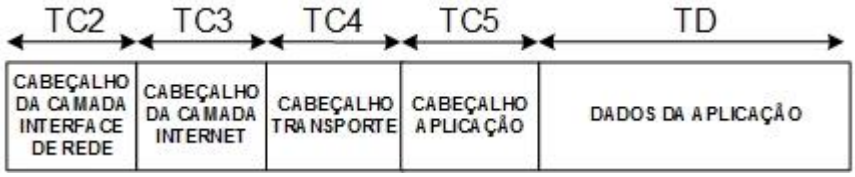
Figura 3.5 - Encapsulamento de Camadas na Arquitetura TCP/IP



Fonte: elaborado pelo autor.

Fica claro que o tamanho de um pacote está diretamente relacionado à quantidade de camadas e o tamanho dos dados da aplicação (*payload)* da última camada. Na Figura 3.6, há uma representação gráfica da composição de um pacote com as cinco camadas do TCP/IP, onde TC e TD é o tamanho da camada e dos dados, respectivamente. Nota-se que cada camada tem seu tamanho definido de acordo com o tipo de mensagem que será enviada. O tamanho de cada cabeçalho é definido de acordo com o protocolo utilizado, mas o tamanho do campo com os dos dados enviados varia de acordo com a finalidade daquele pacote.

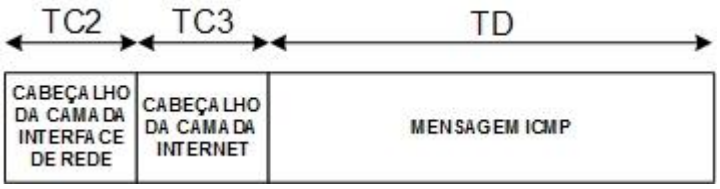
Figura 3.6 - Estrutura Geral de um Pacote na Arquitetura TCP/IP



Fonte: elaborado pelo autor.

O protocolo ICMP [21] é composto pelos campos: “*Type*”, “*Code*”, “*Checksum*”, “*Identifier*”, “*Sequence Number*” e “*Data*”. Dos campos citados, o único campo que o tamanho varia é o campo “Data”. Isso porque, ao enviar uma mensagem de 64 bytes ou 100 bytes, a variação estará no preenchimento desse campo. Na Figura 3.7, o “TD” é indefinido, pois o tamanho desse campo irá variar de acordo com a mensagem, como visto anteriormente. Sendo assim, o tamanho de um pacote ICMP está diretamente relacionado com o tamanho do campo “Mensagem ICMP”, sendo o campo “*data*”, do protocolo ICMP, responsável por essa variação.

Figura 3.7 - Estrutura do Pacote ICMP na Arquitetura TCP/IP

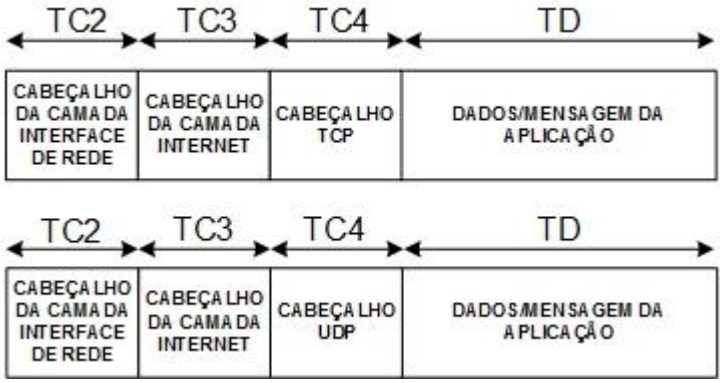


Fonte: elaborado pelo autor.

Já os protocolos TCP [22] e UDP [23] possuem em relação ao protocolo ICMP, uma camada a mais em sua composição, mas assim como no protocolo ICMP o campo “*data*” é responsável pela variação do tamanho dos pacotes destes protocolos, sendo seus cabeçalhos pré-definidos. Tal campo irá variar de acordo com a aplicação. Pacotes de conexões FTP, por exemplo, são diferentes de pacotes de conexões SSH (ambos utilizam o TCP), mas ainda assim a variação estará no campo “*data*” do protocolo TCP. O mesmo se aplica aos pacotes que utilizam o UDP

para realizar uma comunicação. Nesse caso, a variação de um pacote, seja TCP ou UDP, estará no conteúdo do *payload* ou “área de dados da camada de transporte” (Figura 3.8).

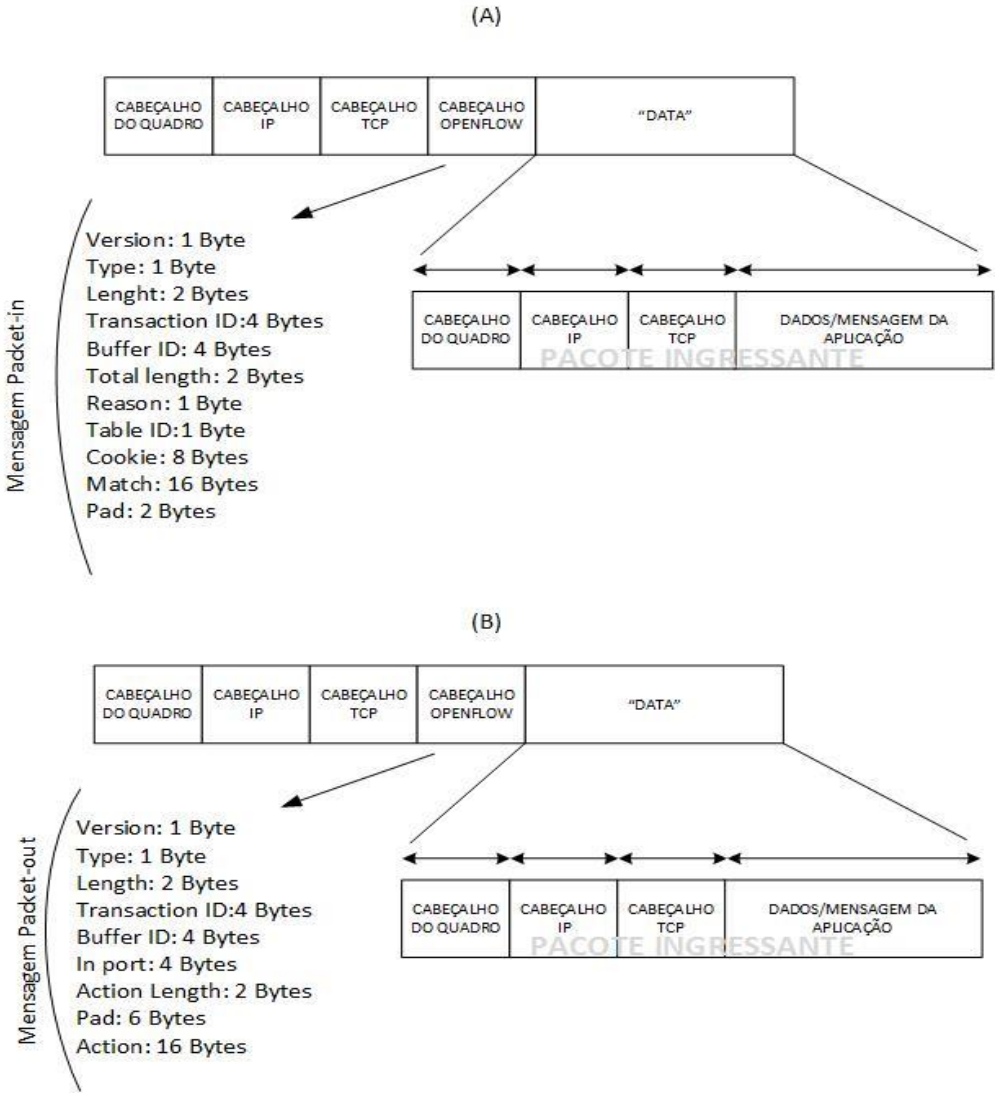
Figura 3.8 - Estrutura dos Pacotes TCP e UDP



Fonte: elaborado pelo autor.

Como a comunicação entre o cliente *OpenFlow* e o controlador é feita através de conexões TCP, as mensagens *Packet-in* e *Packet-out* possuem a mesma estrutura nas camadas 2, 3 e 4. Na camada 5, a diferença de estrutura está nos campos que as compõe, como pode ser visto na Figura 3.9. Nessa mesma figura é possível verificar que uma mensagem *packet-in* tem o tamanho 2 bytes maior que o *packet-out* (caso o controlador não tenha sido alterado para limitar o tamanho dessa mensagem). Isso porque a alteração está nos campos que compõe a estrutura do cabeçalho da mensagem na camada 5 e não no campo “data” (*payload*) dessa.

Figura 3.9 - Estrutura de Mensagens *OpenFlow*. (A) *Packet-in*; (B) *Packet-out*



Fonte: elaborado pelo autor.

Como já dito, o pacote ingressante, quando não tem uma entrada de fluxo correspondente na tabela de fluxo do cliente *OpenFlow*, é encapsulado no *packet-in* a ser enviado ao controlador e retorna no *packet-out* enviado pelo controlador ao cliente *OpenFlow*.

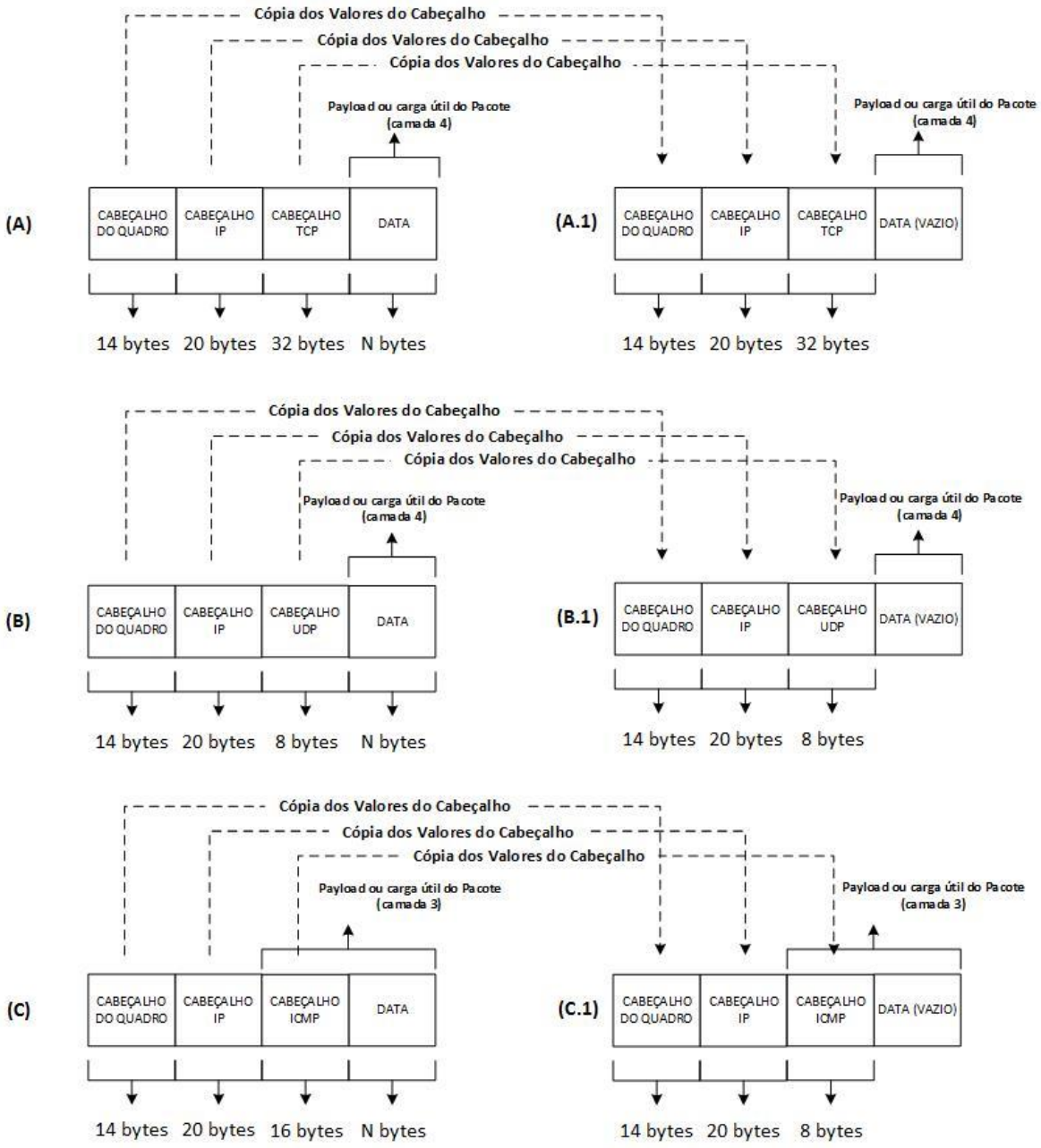
O envio de tais mensagens pode ser configurável, entretanto, apenas no caso do *packet-out*, o tamanho máximo do pacote a ser encapsulado na mensagem pode ser definido. A estrutura de uma mensagem *packet-in* possui tamanho padrão de 42 bytes (como pode ser visto na figura anterior), sendo a variação de tamanho proveniente do pacote a ser encapsulado (com todos os cabeçalhos de cada camada e mais o *payload*).

Tendo a estrutura da mensagem *packet-in* um tamanho pré-definido, o mecanismo de redimensionamento da carga útil do pacote ingressante visa a enviar ao controlador apenas as informações úteis desse pacote ao controlador. As informações presentes no campo "*data*" da última camada do pacote ingressante, geralmente, não são utilizadas pelo controlador para identificar uma entrada fluxo. Já as informações presentes nos cabeçalhos das camadas que o compõe, por sua vez, geralmente são as utilizadas. O envio de uma requisição HTTP, por exemplo, e que não possui previamente uma entrada de fluxo correspondente na tabela de fluxo do cliente *OpenFlow*, terá seu pacote (que contém a requisição completa) encapsulado na mensagem *packet-in* e, ao chegar no controlador, dependendo da configuração ali feita, este irá retornar com a regra definida para esse tipo de serviço. Nesse caso, não importará ao controlador qual a instrução HTTP está dentro do *payload* do pacote, mas, sim, as informações sobre a origem e destino do fluxo que esse pacote irá ter, como: MAC, IP, Protocolo utilizado e Porta.

O controlador pode ser programado para ter uma regra para uma aplicação e uma regra diferente para outra aplicação, sendo a mesma origem e destino (ex.: firewall, balanceadores de carga). Com isso, ao reduzir o pacote ingressante de modo que seja encapsulado apenas as informações presentes no cabeçalho desse, as mensagens *packet-in* e *packet-out* terão, consequentemente, seu tamanho reduzido. Nesse mecanismo, não importa para o controlador se o pacote a ser enviado é uma mensagem ICMP de 64Bytes ou 100Bytes, ou uma mensagem HTTP contendo uma imagem, pois os clientes *OpenFlow* irão encapsular apenas os cabeçalhos deste pacote, desprezando sua carga útil.

Dessa forma, a proposta de redimensionar o tamanho do pacote a ser encapsulado na mensagem *packet-in* não diminuirá apenas a carga para o controlador, mas, também, o tráfego no link que conecta o controlador à infraestrutura de rede [18]. Nesse caso, retiram-se os dados inseridos no *payload* da última camada que compõe a estrutura do pacote ingressante e o insere na mensagem *packet-in* a ser enviada ao controlador. Ao responder à mensagem *packet-in*, o controlador enviará a resposta por meio de uma mensagem *packet-out* contendo o mesmo pacote ingressante, que foi alterado na composição do *packet-in*. A Figura 3.10 ilustra o mecanismo de redimensionamento em pacotes ICMP, TCP e UDP.

Figura 3.10 - Aplicação do Mecanismo de Redimensionamento de Pacote - TCP (A), UDP (B) e ICMP (C)



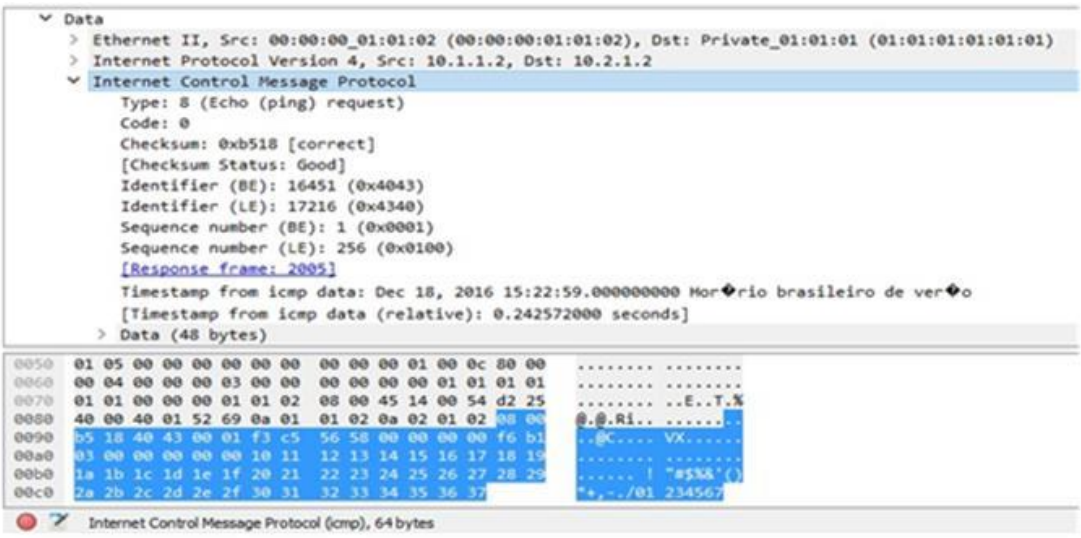
Fonte: elaborado pelo autor.

Em (A), (B) e (C) estão os pacotes da forma como são recebidos pelo cliente *OpenFlow*. Tais pacotes são mantidos da forma como chegam para posterior envio, caso seja necessário a consulta ao controlador. Os pacotes descritos em (A.1), (B.1) e (C.1) representam o pacote que será encapsulado no *packet-in*, e que será enviado ao controlador. Nesse caso, os pacotes serão enviados sem a parte do campo “data” do *payload* da camada 3 (para pacotes ICMP) e sem os campos do *payload* da camada 4 nos casos de ser um pacote TCP ou UDP.

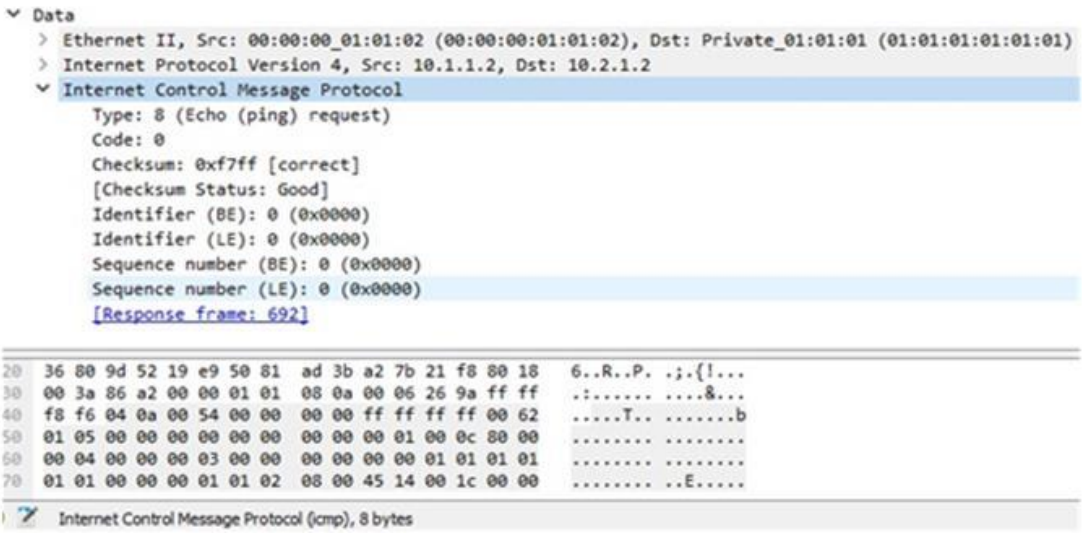
Sendo assim, ao implementar o mecanismo de redimensionamento do pacote, o tráfego do controlador foi capturado para que fosse possível verificar se os pacotes encapsulados em mensagens *packet-in* estavam mantendo as informações dos cabeçalhos das camadas e desprezando os valores do campo “*data”* da última camada. Na Figura 3.11, consta a captura de uma mensagem *packet-in* que tinha um pacote ICMP encapsulado. É possível identificar os campos referentes aos cabeçalhos e que o campo “*data*” não foi mantido no pacote encapsulado.

Figura 3.11 - Captura de Mensagem *Packet-In* Gerada Após o Recebimento de um Pacote ICMP - Comparativo: (A) *Packet-In* Tradicional; (A.1) *Packet-In* Modificado

(A)



(A.1)



Fonte: elaborado pelo autor.

Na figura acima, os pacotes possuem a mesma estrutura nos cabeçalhos que compõe o pacote ICMP. A diferença está na ausência do campo “*data*” do referido pacote. Desprezando o tamanho em *bytes* das demais camadas do pacote (que não são alteradas), em (A) a mensagem ICMP tem o tamanho de 64 bytes, já em (A.1) a

mensagem possui o tamanho de 8 bytes, pois o campo “*data*” está vazio. O mesmo ocorre em pacotes TCP e UDP, em que são mantidos os campos referentes aos cabeçalhos das camadas e desprezados os valores do *payload* que estão no campo “*data*” da camada 4 (quatro).

Ainda, é possível verificar que os dados referentes ao fluxo do referido pacote são enviados em ambos os casos. Os valores de **“MAC Origem e Destino”** e **“IP Origem e Destino”** são enviados sem qualquer modificação. Em pacotes TCP ou UDP também são enviados os dados referentes à **“Porta Origem e Destino”**, presente no cabeçalho da camada 4.

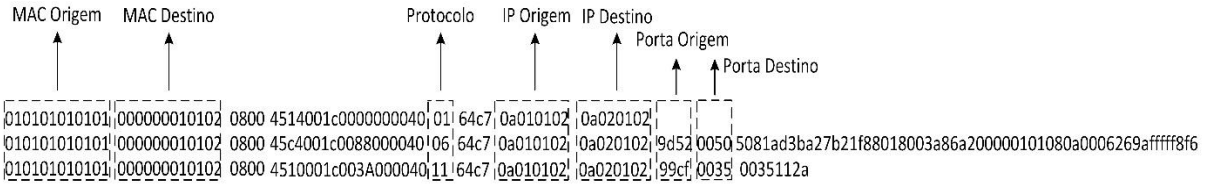
3.1.4 Lógica de Gerenciamento de Pacotes Duplicados

Na lógica de redimensionamento de pacotes, o cliente *OpenFlow* diminui o tamanho da mensagem a ser enviada ao controlador. Contudo, este não evitará o envio de mais de uma mensagem *packet-in* referente a um mesmo fluxo, seja por intervalo de tempo ou pela fragmentação desse pelo tamanho do MTU superior ao configurado no equipamento.

Por esse motivo, o desenvolvimento de um mecanismo que controle esse envio das mensagens *packet-in* fará com que o controlador receba apenas um *packet-in* por fluxo, em um pequeno intervalo de tempo. Esse mecanismo consiste em criar um ***buffer* de *packet-in*** no cliente *OpenFlow*, onde serão armazenadas as informações referentes à mensagem enviada.

As informações escolhidas para identificar um pacote são: MAC Origem, MAC Destino, IP Origem, IP Destino, Protocolo (camada 3), Porta Origem e Porta Destino. Apenas quando se tratar de pacotes TCP e UDP que as informações referentes a portas (origem e destino) serão armazenadas no buffer. Esse procedimento pode ser visualizado na Figura 3.12.

Figura 3.12 - Valores em Hexadecimal de Pacotes ICMP, TCP e UDP



Fonte: elaborado pelo autor.

As informações serão armazenadas no *buffer de packet-in* na ordem de chegada. Sendo assim, caso a ordem seja a da Figura 3.12, com o ICMP request, a solicitação HTTP e a consulta DNS, o *buffer de packet-in* (Figura 3.13) teria o seguinte valor armazenado:

Figura 3.13 - Exemplo de Valores Inseridos na Tabela *Packet-in*

Buffer de Packet-In
1ª Entrada: 010101010101000000010102010a0101020a020102
2ª Entrada: 010101010101000000010102060a0101020a0201029d520050
3ª Entrada: 010101010101000000010102110a0101020a02010299cf0035

Fonte: elaborado pelo autor.

A cada nova mensagem *packet-in*, uma consulta ao *bufffer* de *packet-in* será realizada para evitar o envio de mensagens duplicadas ao controlador. Quando ao consultar o *buffer de packet-in*, se a mensagem *packet-in* correspondente ao fluxo já tenha sido enviada ao controlador e o cliente esteja aguardando a resposta do mesmo, a mensagem não é enviada. Caso contrário, a mensagem *packet-in* é enviada e o *buffer de packet-in* recebe as informações referente àquela entrada. Ao receber a resposta do controlador, referente ao *packet-in* enviado, o cliente *OpenFlow* retira do *buffer* de *packet-in* as informações referentes àquela mensagem. Se no futuro for necessário o envio de mensagem *packet-in* para o controlador, referente ao fluxo que já foi enviado anteriormente (ressalvando-se que no momento é necessário seu envio, seja por ausência de regra ou pela regra pré-estabelicida), a consulta ao *buffer de packet-in* retornará que não há mensagem aguardando resposta, e o processo (de envio da mensagem *packet-in*) iniciará novamente.

É importante ressaltar que durante a comunicação entre cliente *OpenFlow* e controlador interrupções podem ocorrer, seja por queda ou atrasos na resposta do controlador, ou por problemas na rede que impeça que as mensagens *OpenFlow* cheguem a seu respectivo destino. Dessa forma, um contador de consultas foi

implementado, para caso ocorram várias consultas referentes a um mesmo fluxo, o envio de *packet-in* será feito e o contador será zerado para àquele fluxo.

3.1.5 Considerações Importantes da Arquitetura

Há que se falar que o procedimento de redimensionamento do pacote, apenas retira a carga útil do pacote a ser encapsulado no *packet-in*. Isso porque a carga útil do pacote ingressante, que foi armazenado no *buffer* OVS e será enviado posteriormente, não é alterada. Ou seja, apenas o pacote a ser enviado (encapsulado na mensagem *packet-in*) ao controlador tem sua carga útil alterada.

Já o mecanismo que busca evitar a duplicação do envio de mensagens *packet-in* ao controlador foi implementado, inicialmente, com um tamanho máximo de *buffer de packet-in* de 32 entradas. Isso porque, tendo em vista a quantidade de mensagens *packet-in* que são enviadas para o controlador, ao não receber a resposta do controlador, este *buffer* poderia causar um estouro de memória.

Ainda, o mecanismo conta com um contador de consultas ao *buffer de packet-in*. O valor desse contador é incrementado toda vez que uma consulta ao buffer é feita e o pacote é encontrado. Enquanto a resposta do controlador não é recebida o valor vai sendo incrementado. Assim, após algumas consultas a este *buffer* positivas a sua presença, o envio da mensagem *packet-in* ao controlador é permitido, uma vez que a ausência de resposta pode ter ocorrido por fatores externos (perda na transmissão da mensagem ou da resposta, por exemplo).

3.2 Ambiente Experimental – Cenário de Teste

De modo a avaliar o mecanismo *RDP*, foram definidos dois ambientes: (1) cenário utilizando o *Mininet* na arquitetura e topologia proposta em [16]; e (2) cenário utilizando máquinas físicas com OVS instalado. Em ambos cenários, a mudança foi feita diretamente no código fonte do OVS. O controlador, por sua vez, foi implementado em máquina externa ao cenário.

3.2.1 Caso de Uso: Arquitetura SDN MPLS – Cenário I

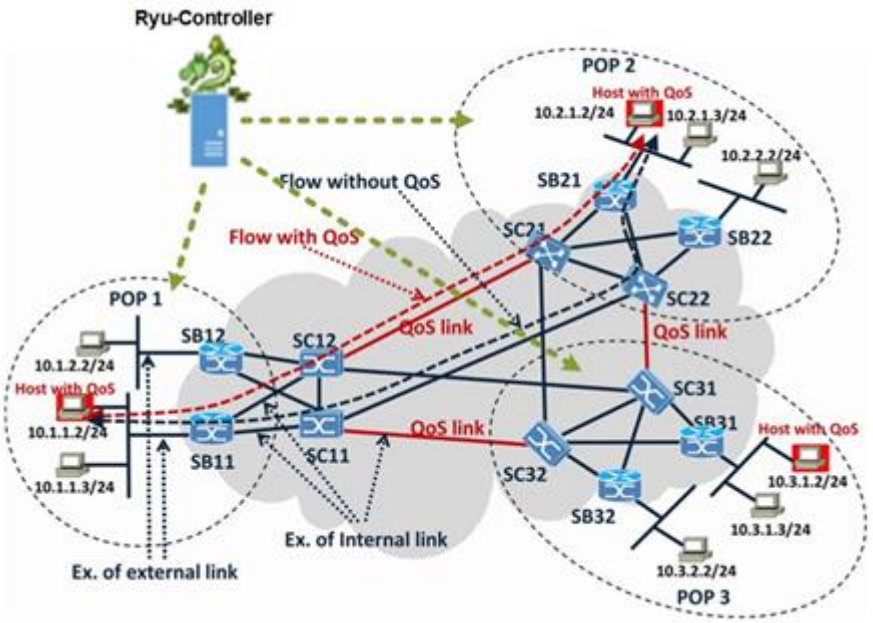
A arquitetura proposta em [16] permite incrementar a robustez da rede sensivelmente, e utiliza-se de regras pré-estabelecidas com o intuito de encaminhar os pacotes de novos fluxos imediatamente. No referido trabalho é implementado um mecanismo que só envia ao controlador mensagem *packet-in* correspondente a

pacotes de novos fluxos com requerimentos de QoS. Os pacotes com estes requerimentos são processados no cliente *OpenFlow* por duas linhas de processamento: encaminhados imediatamente utilizando as regras gerais pré-estabelecidas e, simultaneamente, enviados ao controlador mediante o *packet-in* correspondente. Quando o controlador calcula o caminho ótimo para esse tráfego, constrói as regras específicas correspondentes em todos os nós intervenientes no trajeto ótimo e, dessa forma, o tráfego com requerimentos de QoS é reencaminhado utilizando estas regras especificas.

A mudança proposta em [16] permite desconsiderar os tempos de resposta do controlador e gerar uma arquitetura SDN robusta, capaz de continuar operativa ainda que durante uma queda do controlador. Sendo o único revés a não aplicação de regras para o tráfego com requerimento QoS. Porém, se estas características são combinadas com a diminuição de carga e do tráfego enviados ao controlador propostos neste trabalho, obtemos uma arquitetura que aprimora sensivelmente as arquiteturas OpenFlow padrão atuais.

Diante disso, foi solicitado ao autor, a utilização do cenário implementado por ele na validação do mecanismo RDP. Prontamente, foi liberada pelo autor a utilização do cenário e iniciou-se, assim, o primeiro cenário de validação do mecanismo aqui proposto. Pode-se verificar que o ambiente, conforme Figura 3.14, possui doze switches e nove hosts em sua topologia. E que algumas interligações foram definidas como “*QoS Link*” e outras não (para tráfego sem requerimento QoS).

Figura 3.14 - Topologia do Protótipo Proposto em [16] com Adaptação.



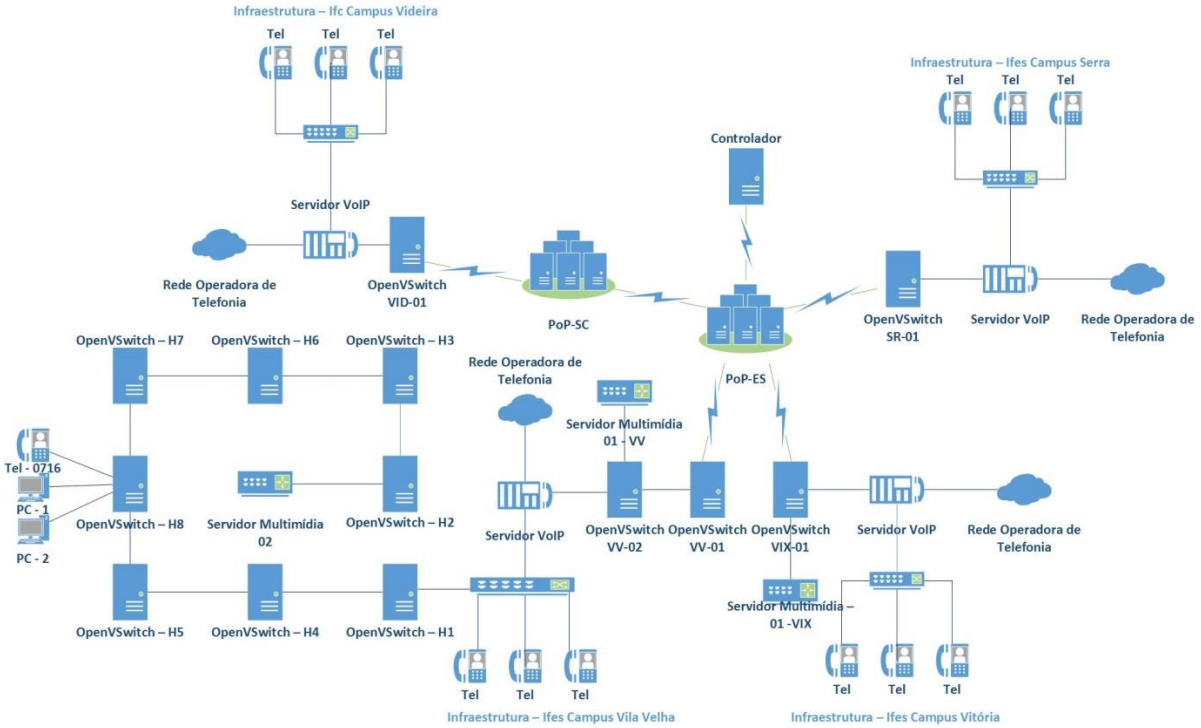
Fonte: retirado de [16] e adaptado pelo autor.

Ao inicializar o cenário, foram necessárias adaptações para a utilização da arquitetura neste trabalho. Por exemplo, o controlador utilizado em [16] estava implementado como externo, mas implementado no mesmo servidor que o *Mininet*. Porém, nesta pesquisa, ficou definido um controlador externo em outro servidor, para que fosse possível mensurar o tráfego enviado à sua interface.

3.2.2 Caso de Uso: Arquitetura Ifes – Cenário II

Na arquitetura, aqui denominada Arquitetura Ifes (Instituto Federal do Espírito Santo), todos os clientes *OpenFlow*, e também o controlador estão instalados em máquinas dedicadas, não sendo necessário utilizar a aplicação Mininet. Foi possível compartilhar diversos serviços de rede tais como: VoIP (*Voice over Internet Protocol*) – por meio da criação de troncos IAX entre os campi Vila Velha, Serra (Ifes) e Videira (Ifc) e também interligação do Telefone IP Interno do Campus Vila Velha com o servidor VoIP desse campus -; acesso à internet e outros serviços disponibilizados na infraestrutura do Ifes, para as estações de trabalho internas do Campus Vila Velha; e interconexão dos geradores de tráfego multimídia – Campus Vila Velha e Vitória do Ifes. A topologia presente no cenário, mostrando cada ponto da rede e suas interconexões, pode ser visualizada na Figura 3.15.

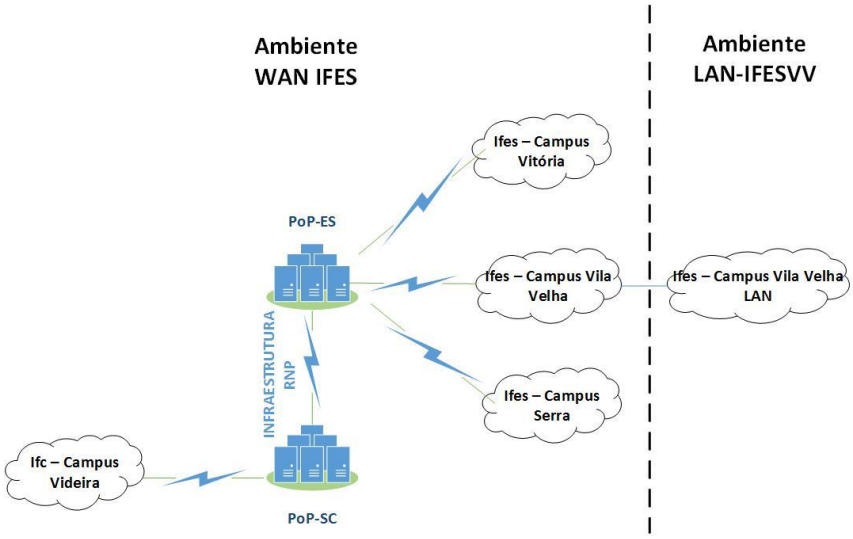
Figura 3.15 - Topologia Detalhada do Cenário II - Ambiente Ifes



Fonte: elaborado pelo autor.

Esse cenário pode ser dividido em dois ambientes: (1) WAN-Ifes; e (2) LAN-Ifes-VV. O WAN-Ifes trata das interconexões WAN entre os campi do Ifes e do Ifc. Tendo em vista que a interconexão desse cenário se dá por meio de *links* de Internet dos locais que o compõe (Figura 3.16), consequentemente, o tráfego desse cenário concorre com outros tráfegos. Já o ambiente LAN-Ifes-VV é um ambiente interno do Ifes Campus Vila Velha, onde os clientes *OpenFlow* que o compõem provêm conectividade para estações de trabalho e telefone IP desse campus, desempenhando, assim, as mesmas funções que os *switches* tradicionais. Trata-se, portanto, de um cenário implementado em um ambiente real e em produção.

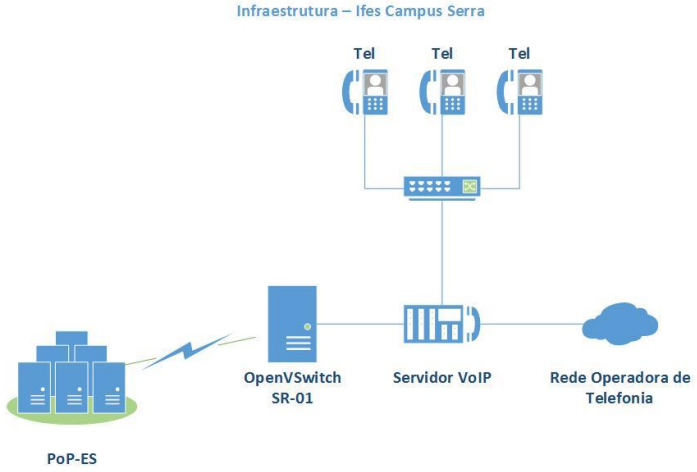
Figura 3.16 - Topologia Resumida do Cenário II



Fonte: elaborado pelo autor.

Os clientes *OpenFlow* que compõem o WAN-Ifes, mesmo estando instalados em locais diferentes (fisicamente e geograficamente) e em máquinas virtuais ou físicas, seguem a mesma topologia de rede (Figura 3.17). Esses locais estão interconectados por meio da infraestrutura provida pelo PoP/RNP do respectivo estado onde se encontra (Espírito Santo ou Santa Catarina).

Figura 3.17 - Topologia do Ambiente WAN-Ifes – Campus

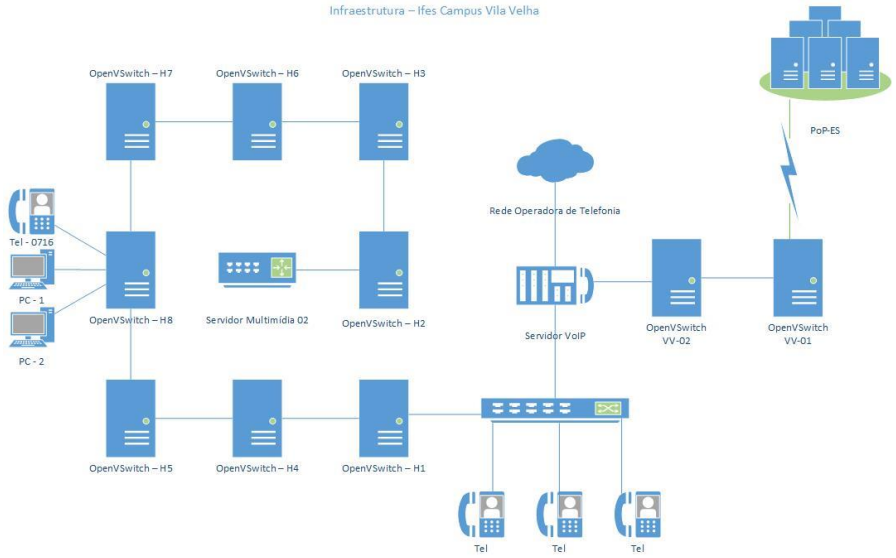


Fonte: elaborado pelo autor.

Sendo o ambiente LAN-Ifes-VV um ambiente interno do Ifes-Campus Vila Velha, sua topologia se difere do ambiente WAN-Ifes (Figura 3.17). Isso porque tal topologia está disposta de modo a atender as estações de trabalho internas e do telefone IP que acessarão os mais diversos tipos de serviços, e não apenas os serviços do ambiente WAN-Ifes. Vale ainda destacar que a topologia

supramencionada desse ambiente (Figura 3.18) conta com 10 (dez) switches virtuais.

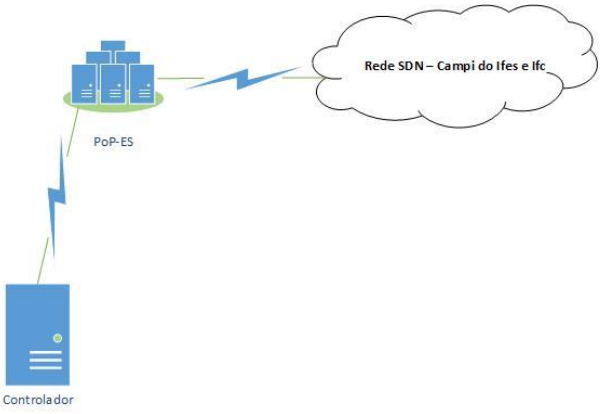
Figura 3.18 - Topologia Ambiente LAN-Ifes-VV



Fonte: elaborado pelo autor.

O trabalho foi proposto à Diretoria de Tecnologia da Informação do Ifes, após a conclusão das validações da implementação do cenário, de modo que fosse prevista a separação do controlador da infraestrutura do Ifes Campus Vila Velha (Figura 3.19). Dessa forma, caso seja deliberada a utilização do ambiente para algum serviço do Ifes, o mesmo já estará disponível para uso.

Figura 3.19 - Topologia do Ambiente WAN-Ifes - Core/Controlador



Fonte: elaborado pelo autor.

Todos os clientes do cenário II não possuem regras pré-estabelecidas e, por isso, todo pacote ingressante que pertença a um fluxo que não esteja na tabela de fluxo, por uma consulta anterior, será encaminhado ao controlador, via *packet-in*. O controlador, por sua vez, possui definida em suas políticas de regras, a liberação de fluxos pertencentes ao cenário (entre os *campi* e institutos) e bloqueio de fluxos

provenientes ou com destino a fluxos não pertencentes a esse cenário. Essa lógica foi escolhida para que a implementação do cenário II não permitisse o acesso externo à rede interna por endereços não conhecidos.

3.3 Infraestrutura dos Cenários de Validação da Proposta

3.3.1 Softwares Utilizados

Com a finalidade de implementar o mecanismo RDP, foi necessário definir os softwares a serem utilizados no ambiente de validação. Tendo em vista a impossibilidade de acesso ao código fonte dos equipamentos dedicados (ex.: HP, Cisco), ficou definida a escolha do *OpenVSwitch*, como sistema de comutação de dados e para exercer a função de controlador, o *Ryu Controller* foi escolhido.

Como os cenários escolhidos para validação da proposta são distintos, no cenário I, foi escolhido o *Mininet*, a fim de que fosse possível criar as instâncias de switches para o ambiente de teste e, no cenário II, as instâncias de switches foram criadas a partir dos equipamentos com o *OpenVSwitch* instalado.

3.3.2 Infraestrutura Utilizada nos Cenários I e II

Para reproduzir os cenários de testes foram utilizados vários tipos de equipamentos com capacidades distintas. Cada cenário tem como objetivo verificar o comportamento do controlador em ambientes distintos. Na infraestrutura do cenário I, o tráfego é limitado ao ambiente reproduzido e, também, aos comandos executados. Já a infraestrutura do cenário II, é o oposto à proposta anterior, uma vez que o tráfego não é controlado e, por isso, há várias possibilidades de fluxos de comunicação. A infraestrutura aqui citada é apenas a que compõe os ambientes utilizados na execução dos testes.

No cenário descrito no tópico 3.2.1, foi utilizado apenas um equipamento que possuía 8GB de memória RAM e processador Intel Core i7 (5500). Nesse equipamento, o sistema Operacional utilizado para reproduzir o ambiente descrito em [16] foi o Ubuntu 14.04 LTS. Os softwares instalados, já descritos no tópico anterior, foram *Mininet* e OVS (para reproduzir a infraestrutura de rede de um SP – *ServiceProvider*) e o *RyuController* (para desempenhar a tarefa do controlador *OpenFlow* da infraestrutura). Sobre o controlador, é importante destacar que ele foi instalado em uma máquina virtual, configurada com 1,5GB de RAM e processador de dois núcleos.

No cenário descrito no tópico 3.2.2, foram utilizados treze equipamentos para serem os clientes *OpenFlow* e o controlador. Tratando-se de um cenário em ambiente real, e por questões de limitação, poucas máquinas possuíam mais de duas interfaces de rede. A Tabela 3.1 contém a configuração do hardware (físico ou virtual) de cada equipamento presente no Ambiente Ifes. Além desses equipamentos, também foi utilizada a infraestrutura de rede de alguns *campi* do Ifes, do IFC e da RNP (Rede Nacional de Pesquisa).

Tabela 3.1 - Configuração de Hardware das Máquinas Utilizadas no Cenário II

Host	**Tipo de Máquina**	**Memória RAM (GB)**	**Interface de Rede**	
			Quantidade	**Velocidade (Mbps)**
CONTROLADOR	Virtual	4	1	10/100/1000
OVS - VV - 01	Física	4	2	10/100/1000
OVS - VV - 02	Virtual	4	2	10/100/1000
OVS - VV - LAN - H1	Física	32	2	10/100/1000
OVS - VV - LAN - H2	Física	32	2	10/100/1000 (1) e 10/100 (1)
OVS - VV - LAN - H3	Física	32	2	10/100/1000 (1) e 10/100 (1)
OVS - VV - LAN - H4	Física	32	2	10/100/1000 (1) e 10/100 (1)
OVS - VV - LAN - H5	Física	32	2	10/100/1000 (1) e 10/100 (1)
OVS - VV - LAN - H6	Física	32	5	10/100/1000 (1) e 10/100 (4)
OVS - VV - LAN - H7	Virtual	4	2	10/100/1000
OVS - VV - LAN - H8	Física	32	5	10/100/1000 (1) e 10/100 (4)
OVS - SR - 01	Virtual	4	2	10/100/1000
OVS - VID - 01	Virtual	4	2	10/100/1000
OVS - VIX - 01	Virtual	4	2	10/100/1000

Fonte: elaborado pelo autor.

Já os links que interligam os campi, e institutos, possuem as seguintes capacidades de conexão (Tabela 3.2):

Tabela 3.2 - Velocidade do Link de Cada Campi do Cenário II

Local	**Capacidade da Conexão (Mbps)**
Ifes – Campus Vila Velha	100
Ifes – Campus Vitória	1000
Ifes – Campus Serra	100
Ifc – Videira	100

Fonte: elaborado pelo autor.

3.3.3 Descrição das Métricas

Tendo em vista que o objetivo do trabalho é diminuir a carga enviada ao controlador, as métricas para mensurar e avaliar o mecanismo RDP serão:

i. Tempo necessário para o processamento de mensagens assíncronas;
ii. Taxa de processamento de mensagens assíncronas;
iii. Mensagens *Error* e *Warning*;
iv. “Saúde” da rede;
v. Consumo de Hardware do controlador.

Dentre as métricas citadas, destacam-se **i** e **ii**. Essas métricas foram definidas a partir da proposta contida no documento, ainda em fase de rascunho, disponível no IETF (*Internet Engineering Task Force*) [31] [32]. Seus objetivos é aferir o desempenho do controlador em um ambiente com clientes *OpenFlow*. Nesse documento, consta o quanto é importante que o controlador responda de maneira rápida às mensagens assíncronas, tendo em vista que elas são utilizadas, também, para solicitar as entradas de fluxo para que um destino seja alcançado.

3.3.3.1 Tempo necessário para processamento de mensagens assíncronas

Essa métrica foi definida para que fosse possível mensurar o tempo que o controlador necessita para processar uma mensagem assíncrona. A mensagem assíncrona utilizada nesse trabalho é o *packet-in*. O cálculo do tempo que o controlador usa para processar uma mensagem assíncrona tomará como base a chegada de uma mensagem *packet-in* até a resposta do controlador a essa mensagem via *packet-out*.

Essa métrica, para ser calculada, necessita que o *packet-out* seja referente a um *packet-in* específico [30]. Com isso, calculou-se apenas o tempo de resposta utilizando *packet-out* que teve seu *packet-in* correspondente. Apesar de ser óbvio, no momento de tabular os dados, houve caso em que não foi encontrado o *packet-in* correspondente a um *packet-out*, além de haver mais de um *packet-out* para um mesmo *packet-in*.

Além de correlacionar os *packet-out* e *packet-in* pelos números de sequência, foi possível correlacioná-los a partir do pacote que estava encapsulado nestes. Ou seja, *packet-out* e *packet-in* com pacotes encapsulados idênticos, foram correlacionados e utilizados nessa métrica.

Os valores utilizados foram obtidos a partir da Equação 1:

Equação 1 - Cálculo do Tempo Necessário para Processar uma Mensagem Assíncrona

$$\boldsymbol{TPMA} = \frac{(\boldsymbol{R1} - \boldsymbol{T1}) + (\boldsymbol{R2} - \boldsymbol{T2}) \ldots (\boldsymbol{Rn} - \boldsymbol{Tn})}{\boldsymbol{Nr}\mathrm{x}}$$

Fonte: retirado de [32]

Onde: **TPMA** é o Tempo de Processamento de Mensagem Assíncrona; **Rn** é o tempo de envio do enésimo *packet-out*; **Tn** é o tempo de recebimento do enésimo *packet-in*; e **Nrx** é a quantidade interações bem-sucedidas.

3.3.3.2 Taxa de processamento de mensagens assíncronas

Além do tempo que o controlador utiliza para responder uma mensagem, há a necessidade de mensurar a quantidade máxima de mensagens que o controlador consegue processar de cada vez. O controlador deve ser capaz de processar o máximo de mensagens *packet-in* recebidas em um intervalo de tempo, possibilitando assim um provisionamento ágil e uma rede flexível. Essas são as principais características de uma infraestrutura SDN. O cálculo dessa taxa é realizado a partir da quantidade de *Packet-in* ingressante e *Packet-out* enviados em um determinado intervalo de tempo.

Os controladores devem ser capazes de responder em um pequeno intervalo de tempo às mensagens assíncronas que receberem dos clientes *OpenFlow*. A quantidade de mensagens assíncronas que um controlador consegue responder por segundo é denominada taxa de processamento de mensagens assíncronas. Essa taxa influencia na métrica do tempo necessário para processamento de uma mensagem assíncrona, pois uma taxa de processamento baixa indica um tempo maior para responder a uma quantidade de mensagens acima dessa.

A partir do momento que o controlador recebe pelo menos uma mensagem assíncrona do cliente *OpenFlow*, ele enviará, como resposta, uma outra mensagem. Por exemplo, um controlador recebe 10 (dez) mensagens assíncronas em um segundo e responde a essas mensagens com outras 10 (dez) mensagens também em um segundo. Nesse caso, pode-se dizer que a taxa de processamento do controlador é de 10 (dez) mensagens assíncronas por segundo naquele instante.

Contudo, caso esse controlador receba 60 (sessenta) mensagens assíncronas em um segundo e responda a todas essas mensagens em 5 (cinco segundos), sendo 15, 12, 18, 10, 5 mensagens a cada segundo, respectivamente, a

taxa de processamento de mensagens assíncronas do controlador é 15m/s, 12m/s; 18m/s; 10m/s e 5m/s (onde m/s é mensagens por segundo).

Dessa forma, é notório que essa taxa pode variar conforme a quantidade de mensagens recebidas varia. Se for calculado o tempo de resposta de uma mensagem assíncrona do exemplo anterior, a primeira terá um tempo de resposta significativamente menor em relação à sexagésima mensagem. A representação da taxa é representada pela Equação 2.

Equação 2 - Cálculo da Taxa de Processamento do Controlador

$$\boldsymbol{TX = n/s}$$

Fonte: retirada de [32].

Onde: **TX** é a Taxa de Processamento de Mensagem Assíncrona, "**n"** é a quantidade de mensagens enviadas pelo controlador e "**s**" equivale a um segundo.

Portanto, é preferível que a taxa seja igual à quantidade de mensagens recebidas, ou seja, o valor de "n" seja igual à quantidade de mensagens recebidas pelo controlador. Para o exemplo anterior, a taxa ideal seria 60 mensagens por segundo. Porém, a taxa não necessariamente está ligada à quantidade de mensagens recebidas, mas sim ao recurso de processamento disponível. É possível que um controlador consiga processar mais de 200 mensagens por segundo, mas também é possível que o controlador não seja capaz de processar 5 (cinco) mensagens por segundo.

Por isso, alterar a estratégia utilizada no envio de mensagens ao controlador, evitando mensagens duplicadas referentes a um mesmo fluxo (no caso de mensagens *packet-in*), proporcionará ao controlador ter a taxa de processamento equivalente à quantidade de mensagens recebidas. Voltando ao exemplo anterior, o controlador recebeu 60 mensagens, mas, destas, 30 eram duplicadas. Sendo assim, se essas mensagens não tivessem sido enviadas, o controlador poderia responder em menor tempo e reduziria o processamento requerido.

3.3.3.3 Mensagens Error e Warning

Essa métrica visa a apresentar um panorama geral do tráfego entre clientes e controlador *OpenFlow*, expondo as anomalias encontradas, como **retransmissão, estouro de buffer**.

A ferramenta “Wireshark”, dentre as opções de análise de tráfego, possui a opção “*Expert Information*” que é um levantamento geral das anomalias encontradas no tráfego capturado.

Dentre os tipos de anomalias, serão expostos: *Error* e Warning. As anomalias do tipo “*Error*” são aquelas que necessitam de maior a atenção, pois pode tratar-se de um pacote malformado, por exemplo. Durante os testes, a anomalia do tipo “*Error*” encontrada foi a de ***Retransmissão***. Essa anomalia é identificada quando um pacote que possui dados tem seu o número de sequência alterado em relação ao anterior e aos próximos, possua dois ou mais ACKs duplicados na direção oposta, referentes ao número de sequência da retransmissão.

As anomalias classificadas como do tipo “*Warning*” são aquelas que podem gerar um impacto no desempenho da rede. As anomalias desse tipo são geradas quando ocorre um erro “incomum”, como um problema de conexão. Há duas anomalias desse tipo e que estão associadas: ***“TCP window specified by the receiver is now completely full”*** e ***“TCP zero window segment”***. A anomalia ***“TCP window specified by the receiver is now completely full”*** ocorre quando a janela TCP definida no receptor encontra-se completamente cheia. E a anomalia ***“TCP Zero Window Segment”*** indica que o cabeçalho está com o valor 0 no campo “*Window size*” e sem as *flags* RST ou FIN definidas, que indicariam o fim de uma conexão. Sendo assim, quando há uma anomalia “*TCP Zero Window Segment*”, após uma ***“TCP window specified by the receiver is now completely full”,*** é possível que, pelo fato do “*receive buffer*” estar cheio, um novo dado só será recebido após a liberação de um espaço no buffer, acarretando no aumento do atraso dos pacotes.

3.3.3.4 Saúde da Rede

A métrica referente à “saúde” da rede visa a apresentar qual o impacto do mecanismo no desempenho da rede, pois a inserção de uma nova funcionalidade nos clientes *OpenFlow* pode acarretar em custos na rede. No cenário I, foram mensuradas as informações sobre o “***jitter***”, a “**latência**” e as **perdas de pacotes** durante os testes entre os hosts. Essa métrica no cenário II correspondeu aos valores relativos ao desempenho dos serviços de rede.

3.3.3.5 Consumo de Hardware do Controlador

As demais métricas definidas buscam mostrar se o objetivo do mecanismo foi alcançado. Por esse motivo, os dados referentes à carga de processamento, utilização de memória RAM e o tráfego na interface de rede (entrada e saída) no controlador foram coletados. Essas informações foram obtidas diretamente do Sistema Operacional do controlador.

3.3.4 Descrição da Coleta de Dados

Para mensurar o desempenho da solução aqui proposta, foram utilizadas ferramentas para coletar os dados referentes às métricas expostas anteriormente. São elas: ***tcpdump*** e ***ps***. E, referente ao tráfego na interface de rede, as informações das interfaces configuradas no sistema foram obtidas diretamente do **/proc/net/dev**.

Com o ***tcpdump*** foi capturado o tráfego de entrada e saída da interface de rede do controlador. Após realizar a captura, foi possível obter as informações para as métricas de tempo necessário para o processamento de mensagens assíncronas, as taxas de perda de pacotes e de pacotes duplicados.

Ao executar a aplicação ***ps***, é possível obter as informações referentes ao consumo do processador e da memória em um determinado momento. Isso se dá por meio da combinação de outras aplicações (***grep*** e ***gawk***) aplicadas à saída da aplicação aqui citada.

Por fim, ao utilizar a aplicação ***gawk*** e ***grep*** no arquivo de saída ***/proc/net/dev*** foi possível obter as informações da interface de rede do controlador, como as quantidades de *bytes* e pacotes que foram enviados e recebidos pelo controlador em um período de tempo.

Para a execução das aplicações citadas, foram desenvolvidos scripts. Por meio dos scripts, definiu-se a rotina de coleta dos dados durante um período de tempo. Em ambos os casos, as aplicações **ovs-vswitchd** e **ryu-manager** ao serem iniciadas, iniciavam também os scripts de coleta dos dados. Também foram armazenados a data e horário da coleta dos dados através da utilização do comando ***date***.

4 AVALIAÇÃO DA PROPOSTA E RESULTADOS OBTIDOS

Neste capítulo será exposto o comportamento de cada cenário durante a execução dos testes. A comparação do cenário, sem e com a proposta deste trabalho, também será apresentada nesse capítulo, bem como os scripts utilizados nos testes, na coleta dos dados e outras informações relevantes ao cenário.

4.1 Definição do Escopo de Avaliação

De modo a avaliar o mecanismo proposto, foram estabelecidos roteiros de execução de testes para que fossem geradas as solicitações de fluxo ao controlador, por parte dos clientes *OpenFlow*, em cada um dos cenários. Esses roteiros foram definidos de acordo com o cenário, tendo em vista sua particularidade.

É importante salientar que o número de interações, as topologias, cenários foram definidos seguindo a orientação contida nos esboços do IETF ([31] e [32]). Foram considerados, ainda, apenas os dados de testes concluídos, conforme orientação do esboço. Definiu-se de maneira aleatória o valor sete como quantidade máxima de consultas do contador definido no mecanismo de deduplicação de pacotes que compõe a estratégia proposta neste trabalho (Figura 3.3).

A escolha da aplicação presente no roteiro levou em consideração a composição do tráfego escolhido. Dessa forma, foi possível, por exemplo, definir o tamanho do pacote que trafega no cenário no momento da execução do roteiro, cuja execução no ambiente foi executada sem e com o mecanismo RDP.

Tendo em vista que o cenário I é proveniente da utilização da aplicação *Mininet*, foi necessário utilizar aplicações que criassem o tráfego de rede. A aplicação **ping**, por exemplo, proporcionou pacotes **icmp** com vários tamanhos. Também foi possível, por meio dessa aplicação, definir o intervalo de tempo que cada envio seria executado.

Além dessa ferramenta, onde os dados foram pré-definidos, no cenário II, há vários tipos de protocolos, uma vez que se trata de um cenário em um ambiente real e em produção. Nesse cenário, os dados coletados são provenientes de requisições VoIP, requisições DNS, e demais protocolos e serviços de redes.

Com isso, foi possível verificar o comportamento do ambiente sem e com a modificação em cenários diferentes. No primeiro cenário, o objetivo foi de averiguar

o comportamento do controlador com várias requisições simultâneas e com tamanhos diversos em um período de tempo pré-definido. Já no segundo cenário, como ele se comporta em uma infraestrutura real e em produção, durante um período de tempo maior e com requisições sendo enviadas constantemente. Nesse cenário, a resposta do controlador iria liberar ou não o envio do pacote ingressante.

4.1.1 Cenário I

No cenário I, como já exposto no tópico 3.2.1, o ambiente com os clientes *OpenFlow* foi reproduzido utilizando o *Mininet* (segundo a arquitetura indicada em [16]). O controlador foi implementado em um servidor externo a esse, simplificando a captura dos pacotes enviados ao controlador.

Nesse cenário, são as estações em vermelho (ver Figura 3.14) que geram tráfego com requerimentos de QoS. Os switches SB11, SB21 e SB31, conectados a esses hosts, geram mensagens packet-in ao controlador, visto que só pacotes de novos fluxos com requerimentos de QoS geram consultas ao controlador.

Para que o mecanismo fosse avaliado da maneira eficaz, ou seja, quando o controlador responde ao fluxo solicitado, os comandos do roteiro desse cenário se basearam apenas nos hosts que estavam conectados nos switches indicados.

As informações básicas de cada comando presente no roteiro, conforme a Tabela 4.1, expõem que vários tamanhos de pacotes trafegaram. Torna-se possível, então, verificar o comportamento do controlador, e também dos switches *OpenFlow*, nesse ambiente de acordo com o tamanho do pacote.

Tabela 4.1 - Pacote ICMP - Roteiro Cenário I

H11 <-> H21 / H11 <-> H31 / H21 <-> H31			
TIPO DE PACOTE	**QUANTIDADE**	**INTERVALO (s)**	**TAMANHO (Bytes)**
ICMP	1000	0,01	56
			512
			1024
			1512
			5000

Fonte: elaborado pelo autor.

O cenário I, por ser um cenário emulado, necessita da interação ativa para que exista um tráfego. O **ping** foi escolhido como a ferramenta a ser utilizada nesse ambiente, por ser uma ferramenta simples. Por meio de suas mensagens de saída,

é possível saber se a mensagem enviada conseguiu alcançar o destinatário com sucesso e qual a “saúde” da rede (*jitter*, latência e quantidade de pacotes perdidos).

Para a execução desse roteiro, foram criados scripts em cada host e iniciada a execução desse roteiro. Cada comando foi repetido vinte vezes em cada host e seu resultado foi armazenado em um arquivo de texto (como pode ser visto no Quadro 4.1). Como a proposta de [16] baseia-se em carregar os clientes OpenFlow com regras gerais, em que as alterações ocorrem apenas em alguns casos específicos, executou-se, juntamente com os scripts dos testes, um script que reinicializava as tabelas de fluxos dos clientes OpenFlow, de forma que apenas as regras iniciais estivessem presentes antes de uma nova execução de comando de envio de pacotes ICMP (descrito nos quadros abaixo).

Quadro 4.1 - Script de Teste I (56 bytes) – Host H11

```
#!/bin/bash

while ((20 > i))
do
ping 10.2.1.2 -i 0.01 -c 1000 -q >>
ping_h21.txt
ping 10.3.1.2 -i 0.01 -c 1000 -q >>
ping_h31.txt
let i++
done
```

Fonte: elaborado pelo autor.

A rotina apresentada acima foi executada cinco vezes para cada host sendo cada uma com tamanho de pacote diferente. Além do envio de pacotes de tamanho 56 bytes (quadro anterior), foram enviados pacotes de tamanho 512, 1024, 1512 e 5000 bytes, como pode ser visto nos itens (A), (B), (C) e (D) do Quadro 4.2.

Quadro 4.2 - Scripts de Testes II, III, IV e V - Cenário I - Host h11

(A)

```
#!/bin/bash

while ((20 > i))
do
ping 10.2.1.2 -i 0.01 -s 512 -c 1000 -q >>
ping_h21.txt
ping 10.3.1.2 -i 0.01 -s 512 -c 1000 -q >>
ping_h31.txt
let i++
done
```

(B)

```
#!/bin/bash

while ((20 > i))
do
ping 10.2.1.2 -s 1024 -i 0.01 -c 1000 -q >>
ping_h21.txt
ping 10.3.1.2 -s 1024 -i 0.01 -c 1000 -q >>
ping_h31.txt
let i++
done
```

(C)

```
#!/bin/bash

while ((20 > i))
do
ping 10.2.1.2 -s 1512 -i 0.01 -c 1000 -q >>
ping_h21.txt
ping 10.3.1.2 -s 1512 -i 0.01 -c 1000 -q >>
ping_h31.txt
let i++
done
```

(D)

```
#!/bin/bash

while ((20 > i))
do
ping 10.2.1.2 -s 5000 -i 0.01 -c 1000 -q >>
ping_h21.txt
ping 10.3.1.2 -s 5000 -i 0.01 -c 1000 -q >>
ping_h31.txt
let i++
done
```

Fonte: elaborado pelo autor.

Os scripts descritos anteriormente foram executados em todos os hosts (H11, H21 e H31) e repetido em cada ambiente (sem e com o mecanismo RDP). Além dos scripts de teste, foram executados scripts de monitoramento nos switches e no controlador. Os comandos presentes nos scripts foram definidos com o objetivo de obter os dados para mensurar as métricas definidas.

Para mensurar o impacto do mecanismo RDP nos switches, foi executado um script com comandos para armazenar as informações da aplicação ***ovs-vswitchd***. Essa aplicação é executada assim que o ambiente de teste é iniciado pelo ***Mininet*** Dessa forma, ao iniciar o ambiente, o script era iniciado em paralelo. No Quadro 4.3 está o script de monitoramento do OVS.

Quadro 4.3 - Script Monitoramento - Cenário I – OVS

```
#!/bin/bash

while ((100000 > i))
do
  let i++
  if (($i%30 < 1))||(($i < 2))
  then
    echo `date` `./monitoramento.sh` >> log-
monitoramento.txt
  fi
  sleep 1
done
```

```
#!/bin/bash

sudo ps aux | grep "ovs-vswitchd --pidfile --detach" |
 gawk '{print $3,$4,$5,$6,$11}' | grep ovs-vswitchd
```

Fonte: elaborado pelo autor.

No controlador, as informações foram coletadas a cada segundo durante a realização dos testes. Executaram-se os comandos de modo que as informações da aplicação "***Ryu-Manager***" fossem armazenadas em um arquivo de texto. O script executado no controlador durante os testes pode ser visualizado no Quadro 4.4.

Quadro 4.4 - Script de Monitoramento – Cenário I – Controlador

```
#!/bin/bash

while ((100000 > i))
do
./mrtgstats-cpu.sh >> log-cpu-mem.txt
./mrtgstats-net.sh >> log-net.txt
sleep 1
let i++
done
```

```
#!/bin/bash

ps aux | grep "ryu-manager" | gawk '{print $3,$4,$5,$6,$11}' | grep python
```

```
#!/bin/sh

echo `date` - `cat /proc/net/dev | grep eth1 | gawk '{print $2,$3,$10,$11}'`
```

Fonte: elaborado pelo autor.

Além do monitoramento do consumo de hardware, coletou-se todo o tráfego de entrada e saída da interface de rede do controlador. Por meio dessa coleta, calculou-se o tempo de resposta das mensagens assíncronas *packet-in*. Para que toda comunicação entre o cliente *OpenFlow* fosse capturada, o comando de captura do tráfego (Quadro 4.5) foi executado momentos antes de inicializar a aplicação "***ryu-manager***" na máquina virtual onde estava o controlador da rede.

Quadro 4.5 - Monitoramento (Captura de Tráfego) - Cenário I – Controlador

```
$ tcpdump -w trafego_XX-YY-ZZZ.pcap -i eth0 -P inout
```

Fonte: elaborado pelo autor.

4.1.2 Cenário II

Conforme descrito em 3.2.2, o cenário II é composto por vários clientes *OpenFlow* distribuídos em locais diferentes. Os equipamentos instalados no Ifes Campus Vila Velha foram aproveitados para atender a algumas demandas internas do Campus como: conectividade das máquinas de pesquisa do laboratório de química [33] e [34]; conectividade externa do servidor VoIP do Ifes Campus Vila Velha com outros Campi do Ifes e com outro Instituto (IFC); interligação dos geradores de tráfego multimídia utilizado por pesquisadores do Ifes Campus Vitória [35]; e interconexão de telefones VoIP e Impressora da Coordenação de Tecnologia da Informação do Ifes Campus Vila Velha com a infraestrutura de rede do mesmo.

Esse cenário, por estar inserido em uma infraestrutura real e em produção, necessitou que fosse desenvolvida uma aplicação específica para o controlador.

Essa aplicação permite apenas que máquinas previamente cadastradas podem ter acesso à infraestrutura SDN desse cenário. Isso porque os servidores VoIP dos *campi* do Ifes e do Ifc utilizam IPs públicos, como os servidores e geradores de tráfego multimídia.

Dessa forma, para que outros IPs externos não tivessem acesso a essa infraestrutura, a aplicação desenvolvida no controlador de rede permitiu apenas acesso aos IPs cadastrados, sendo os demais bloqueados. Essa estrutura está baseada em quatro funções (ou subprogramas) dentro da aplicação principal, aqui denominada "**Controle_Ifes.py**". A primeira função – "***buscaIPRange(...)"*** - é responsável por realizar uma busca na lista dos endereços IPv4 cadastrados e retornar 0 (zero) caso o IP não seja encontrado ou 1 (um) caso o IP esteja na lista (

Quadro 4.6).

Quadro 4.6 - Trecho de Código da Aplicação do Controlador do Ifes - Cenário II

```
def busca_ip_range(self, ipv4_mod):
    for i in range(0, 80):
        if ipv4_mod == range_ip[i]:
            return 1
    return 0

def construtor_flow_ipv4(self, datapath, message, eth, pkt):
    ...
    if self.busca_ip_range(ipv4src) and self.busca_ip_range(ipv4dst):
        ...
        if i == 2:
            self.regras_switch_vix_01(parser, match, message, eth, ofp, i, port_in,
ipv4src, ipv4dst)
        ...
    else:
        print "******* Regra Bloqueio *********"
        print ("Packet-in Switch id: ", i)
        print ("Ip Origem: ", ipv4src, " Ip Destino: ", ipv4dst)
        print ("Porta de entrada: ", port_in)
        print "******* Regra Bloqueio *********"
        self.construtor_flow_block(datapath_sw[i], message, eth, pkt)
```

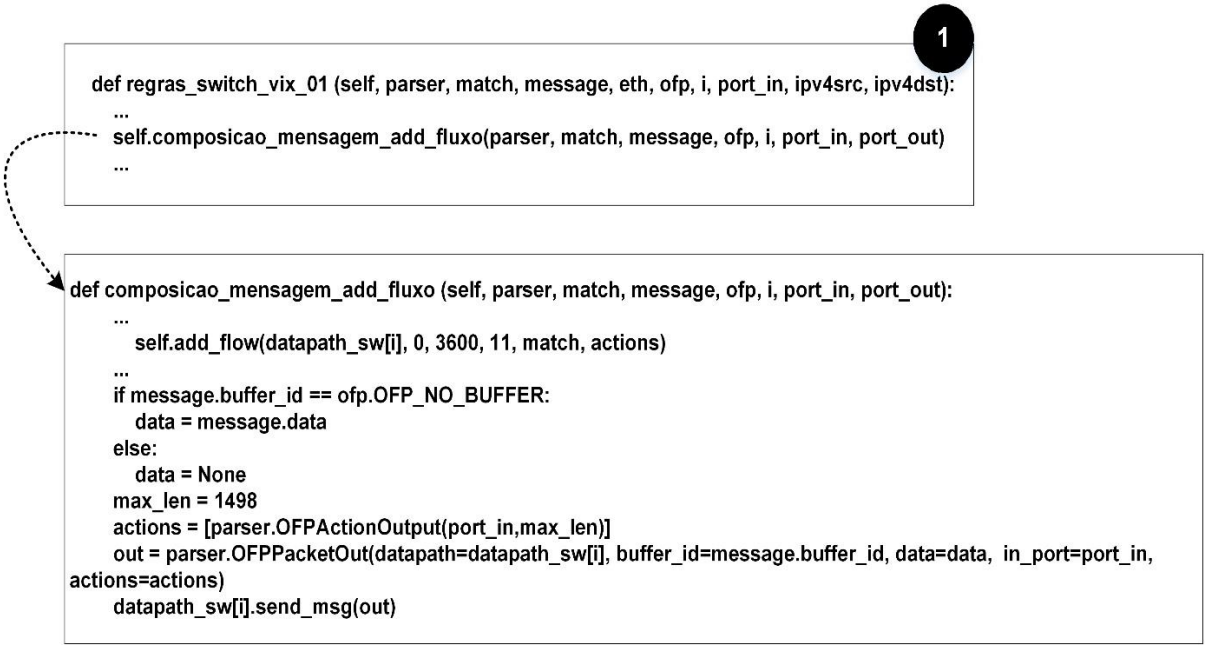
1

2

Fonte: elaborado pelo autor.

Se a mensagem *packet-in* for proveniente do cliente *OpenFlow* instalado no Ifes Campus Vitória, e a função "***busca_ip_range(...)***" retorne um, a função "***regras_switch_vix_01(...)***" será executada e o fluxo será liberado após executar as funções subsequentes (Quadro 4.7).

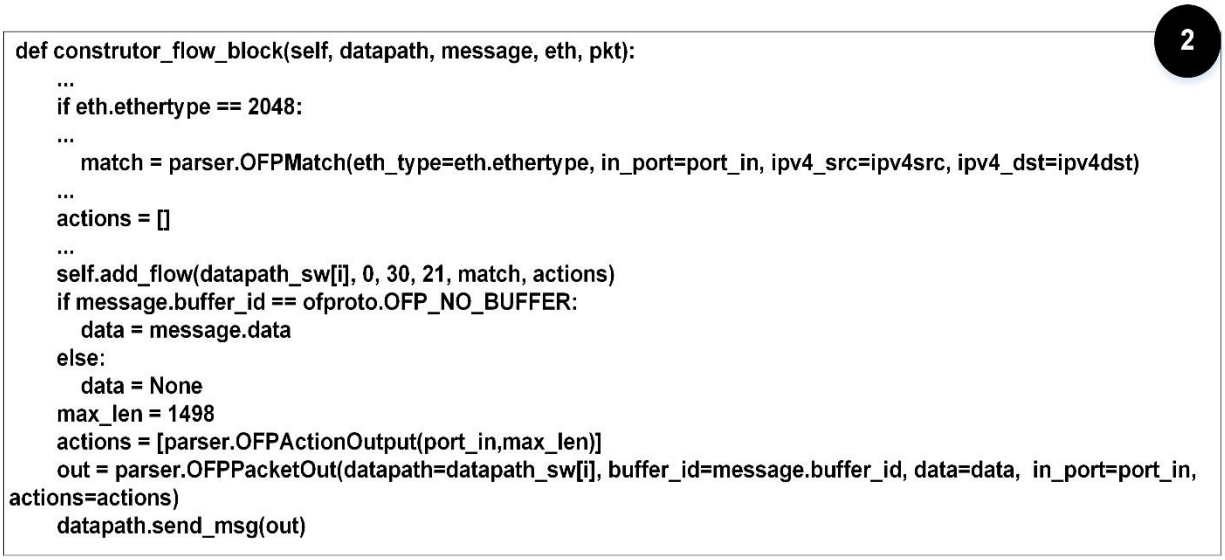
Quadro 4.7 - Trecho de Código da Aplicação do Controlador do Ifes - Cenário II - Liberando Fluxo



Fonte: elaborado pelo autor.

Contudo, caso o retorno da busca na lista de IPs seja 0 (zero), a função "***construtor_flow_block(...)***" será executada e, dessa vez, o controlador enviará uma mensagem para o cliente *OpenFlow* bloquear o tráfego para o fluxo presente no pacote encapsulado no *packet-in* (Quadro 4.8).

Quadro 4.8 - Trecho de Código da Aplicação do Controlador do Ifes - Cenário II - Bloqueando Fluxo



Fonte: elaborado pelo autor.

Em ambos os casos, as mensagens a serem enviadas pelo controlador, seja para liberar ou bloquear o tráfego para o fluxo presente no pacote, serão enviadas ao cliente *OpenFlow* pela função "***add_flow(...)***". Essa função está presente no

código da aplicação de demonstração do Ryu, chamada “***simple_switch_13.py***”, e pode ser visualizada no Quadro 4.9.

Quadro 4.9 - Trecho de Código da Aplicação do Controlador do Ifes - Cenário II - Adicionando a Regra no Cliente *OpenFlow*

```
def add_flow(self, datapath, table, idle, priority, match, actions):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS, actions)]
    mod = parser.OFPFlowMod(datapath=datapath, table_id=table, idle_timeout=idle,
                            priority=priority, match=match, instructions=inst)
```

Fonte: elaborado pelo autor.

Além dos clientes *OpenFlow* que possuem acesso externo, há também os clientes *OpenFlow* internos do Campus Vila Velha. Esses clientes foram gerenciados pelo mesmo Controlador. Sendo assim, a aplicação “Controle_Ifes.py” também foi desenvolvida para esse cenário interno. Nesse caso, se aplica a mesma lógica já citada: o acesso à infraestrutura SDN é liberado apenas para os IPs cadastrados.

Essa aplicação foi desenvolvida com o intuito de não alterar a infraestrutura existente nos locais onde foram instalados os clientes *OpenFlow*, ou seja, não foi necessário implementar novas regras no firewall de cada localidade para proteger a infraestrutura. Tampouco foi necessária a implementação de regras de NAT ou de roteamento. E, ainda, após a conclusão deste trabalho, os clientes *OpenFlow* instalados, bem como a aplicação desenvolvida no controlador, ficarão como produto para utilização do Ifes. É importante salientar que o projeto foi concebido de forma que o controlador possa ser deslocado para a Reitoria do Ifes, sem qualquer prejuízo à infraestrutura já estabelecida, possibilitando, assim, a expansão dessa infraestrutura para outros *campi* do Ifes.

Além da aplicação desenvolvida, foram implementados os scripts de monitoramento em todos os equipamentos. Os scripts são os mesmos apresentados no item anterior (referente ao cenário I), como também a captura do tráfego no controlador.

4.2 Resultados Obtidos

Realizados os testes, os resultados obtidos foram tabelados para que as métricas descritas no tópico 3.3.3 fossem mensuradas e representadas de maneira gráfica. É importante ressaltar que em ambos os cenários foi utilizada a mesma

versão do OVS e do *Ryu-Controller*, sendo diferente apenas sua implementação (no *Mininet* e em máquinas dedicadas).

A representação gráfica de cada métrica utiliza a descrição “Sem Modificação”, quando o OVS estava em sua versão padrão e “Com Modificação”, já com o mecanismo RDP implementado.

A métrica de tempo necessário para o processamento e a taxa de processamento de mensagens assíncrona foram calculadas utilizando apenas pacotes correspondentes. Pacotes correspondentes são aqueles que possuem, em seu cabeçalho TCP, o valor de referência do que o provocou. Ou seja, um *packet-out* possui no cabeçalho TCP o valor do campo referência (*tcp*.*synack*) preenchido com o valor do número de sequência do *packet-in* recebido (*tcp*.*ack*). Sendo assim, apenas os pacotes que possuíam um *packet-out* e *packet-in* correspondente foram utilizados para o cálculo.

4.2.1 Cenário I

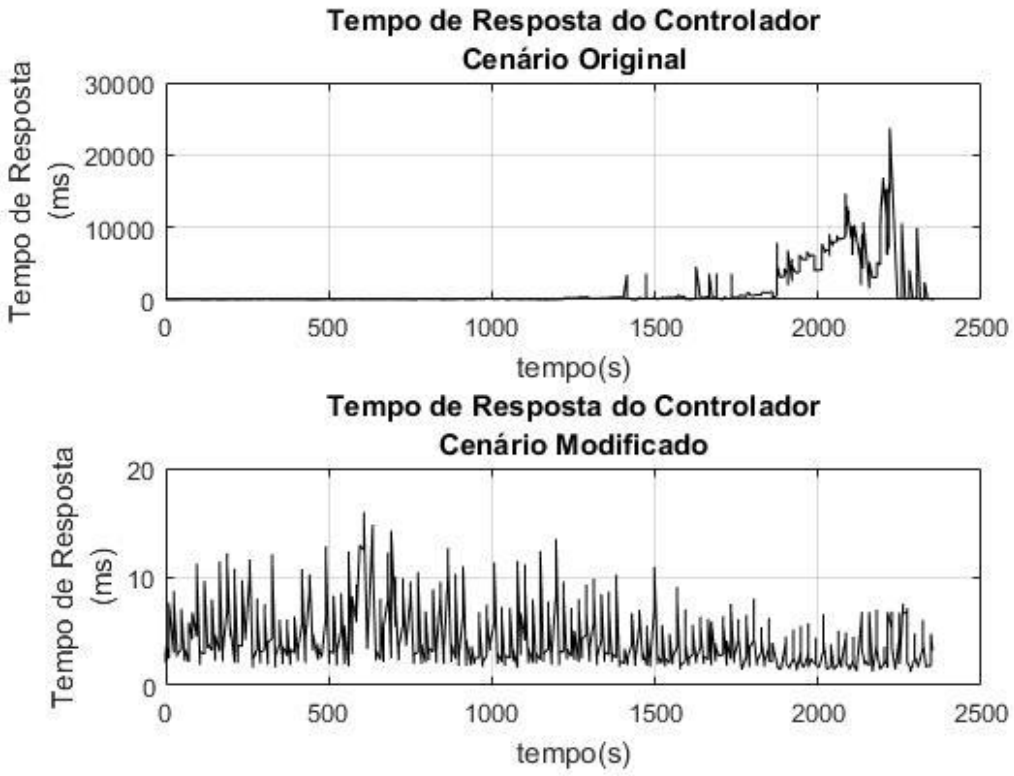
As métricas apresentadas foram obtidas a partir da execução de três testes em cada ambiente. Esses testes foram realizados utilizando vários tamanhos de pacotes, como descrito em 4.1.1 para que fosse possível mensurar o comportamento do ambiente com diferentes tamanhos e quantidade de pacotes. Concluídos os testes, foi feita a média aritmética dos resultados em cada ambiente comparando-os posteriormente.

4.2.1.1 Tempo necessário para o processamento de mensagens assíncronas

Durante a execução dos testes, foi feita a captura dos pacotes recebidos e enviados pelo controlador. E, a partir dos dados obtidos dessa captura, foi possível mensurar o tempo necessário para o processamento das mensagens assíncronas pelo controlador em cada ambiente.

O **TPMA** obtido nesse cenário sem e com o mecanismo RDP, definido na estratégia proposta neste trabalho, foi 5,87s e 0,01s, respectivamente. É notória a diminuição no tempo necessário para o processamento de mensagens assíncronas quando a estratégia do mecanismo RDP é adotada. E após calcular os valores de cada interação bem-sucedida (utilizando a Equação 1), é possível verificar, nos gráficos presentes na Figura 4.1, o comportamento do ambiente sem e com o mecanismo RDP implementado durante a execução dos testes.

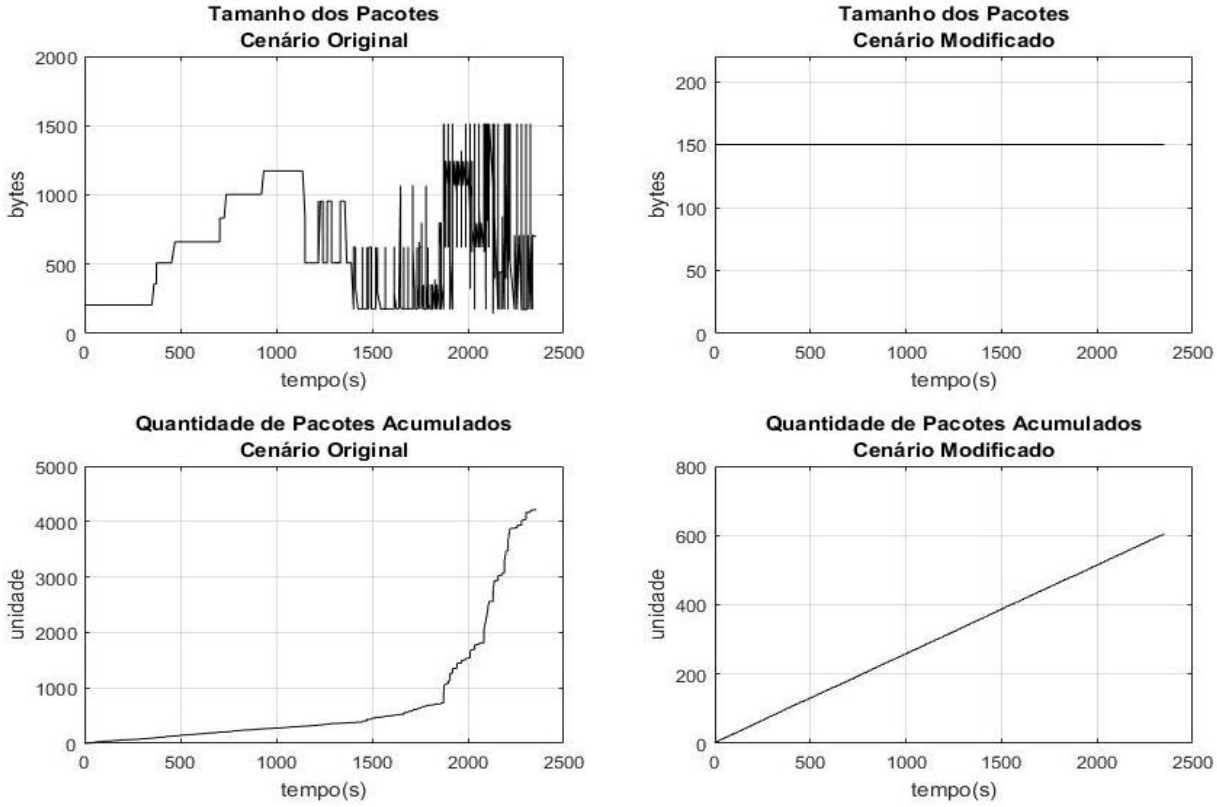
Figura 4.1 - Tempo Necessário para o Processamento de Mensagens Assíncronas *Packet-In*



Fonte: elaborado pelo autor.

O tempo de resposta no cenário modificado foi, ao longo do teste, se mantendo dentro de um padrão. Já no cenário original, o valor também se manteve em um padrão, mas, apenas, até a metade da execução dos testes. Tal comportamento deve-se ao fato da alteração do tamanho dos pacotes e sua quantidade, como pode ser visto na Figura 4.2.

Figura 4.2 - Tamanho e Quantidade Acumulada de Pacotes Enviados ao Controlador Durante a Execução dos Testes no Cenário I



Fonte: elaborado pelo autor.

A diferença no tempo necessário para processar uma mensagem *packet-in* no cenário, sem e com o mecanismo RDP, pode ser compreendida quando se analisa os gráficos presentes na Figura 4.2. No cenário sem o mecanismo, a variação na quantidade e no tamanho dos pacotes enviados ocorrem no mesmo momento que o tempo necessário para processar as mensagens assíncronas aumenta.

Já no cenário com o mecanismo RDP, o tempo necessário para processar as mensagens assíncronas não sofre altas variações durante a execução do teste, assim como a quantidade e o tamanho das mensagens enviadas. O tamanho do pacote que se mantém inalterado no decorrer dos testes em 150 bytes. Desses, 108 bytes são da mensagem packet-in e os cabeçalhos de cada camada e mais 42 bytes referente ao pacote encapsulado na mensagem packet-in. Nesses gráficos é possível ver o resultado de ambas as características do mecanismo RDP: retirada do campo “*data*” presente no *payload* e o controle deduplicação de *packet-in*.

4.2.1.2 Taxa de Processamento de Mensagens Assíncronas

Durante a execução dos testes, a taxa de processamento de mensagens assíncronas (obtida por meio da Equação 2), em média, foi de nove mensagens por

segundo no cenário sem o mecanismo. O valor máximo dessa taxa foi de 50 mensagens por segundo, sendo que o valor máximo de mensagens assíncronas (*packet-in*), recebidas em um segundo, foi de 142 mensagens. Sendo a taxa máxima alcançada de 50 mensagens por segundo e tendo sido recebido durante a execução do teste 142 mensagens em um segundo, é fato que a taxa máxima que o controlador alcançou não atendeu à demanda da rede.

O cenário com o mecanismo RDP teve uma taxa de três mensagens por segundo, em média. Contudo, o valor máximo de mensagens recebidas foi de quatro mensagens por segundo, sendo o valor máximo da taxa de processamento de quatro mensagens por segundo.

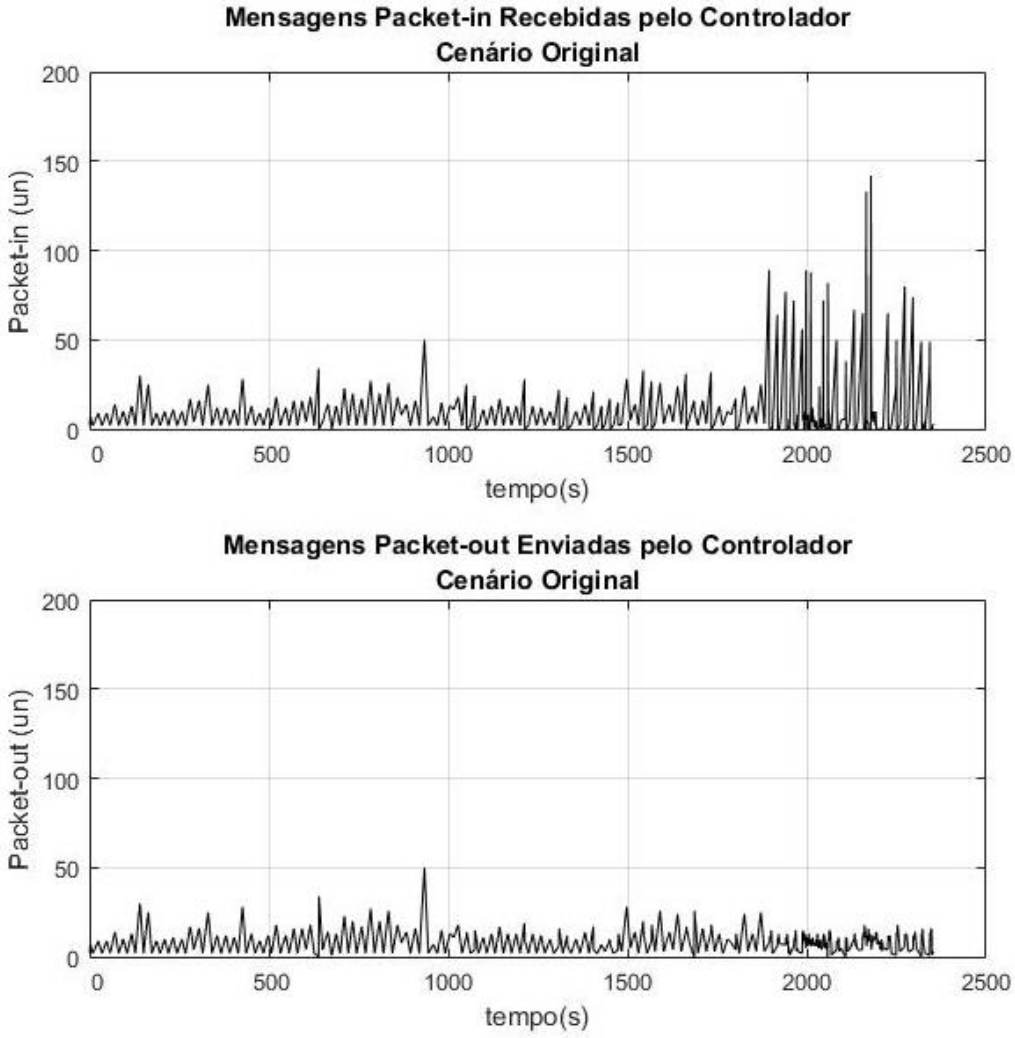
A taxa de processamento de mensagens assíncronas está relacionada à quantidade de mensagens recebidas em um segundo, pois, ao receber várias mensagens em um segundo, o controlador deve ser capaz de responder à mesma quantidade também em um segundo. Ou seja, salvo se alterar a capacidade de processamento do controlador, a forma de se ter um equilíbrio na quantidade de resposta com a quantidade de requisições está em alterar a estratégia utilizada no envio de mensagens no cliente. Nesse caso, o mecanismo RDP evitou que mensagens sucessivas referentes a um mesmo fluxo fossem enviadas ao controlador, diminuindo, assim, a quantidade de mensagens recebidas. Isso resulta em um equilíbrio na quantidade de mensagens recebidas com a taxa de processamento dessas mensagens.

Em ambientes tradicionais, se o Controlador passa um tempo longo recebendo mais mensagens que ele consegue processar, isso resultará em uma sobrecarga e estouro de buffer do mesmo. Por meio do emprego do mecanismo RDP, é mais difícil disso ocorrer.

O tempo necessário para processar uma mensagem assíncrona (abordado anteriormente) é influenciado pela taxa de processamento de mensagens assíncronas, pois uma taxa baixa com o recebimento alto acarreta em um tempo maior para processar todas as informações. No cenário I, realizados os testes, verificou-se que o tempo necessário para processamento de mensagens assíncronas foi maior após o milésimo quingentésimo segundo (1500’’). Dois fatores destacam nesse mesmo momento: aumento do tamanho e da quantidade de pacotes enviados ao controlador.

Para entender melhor o desempenho do ambiente sem o mecanismo no cenário I, os gráficos apresentados na Figura 4.3 possuem o quantitativo de *packet-in* ingressante e de *packet-out* enviado durante a execução dos testes.

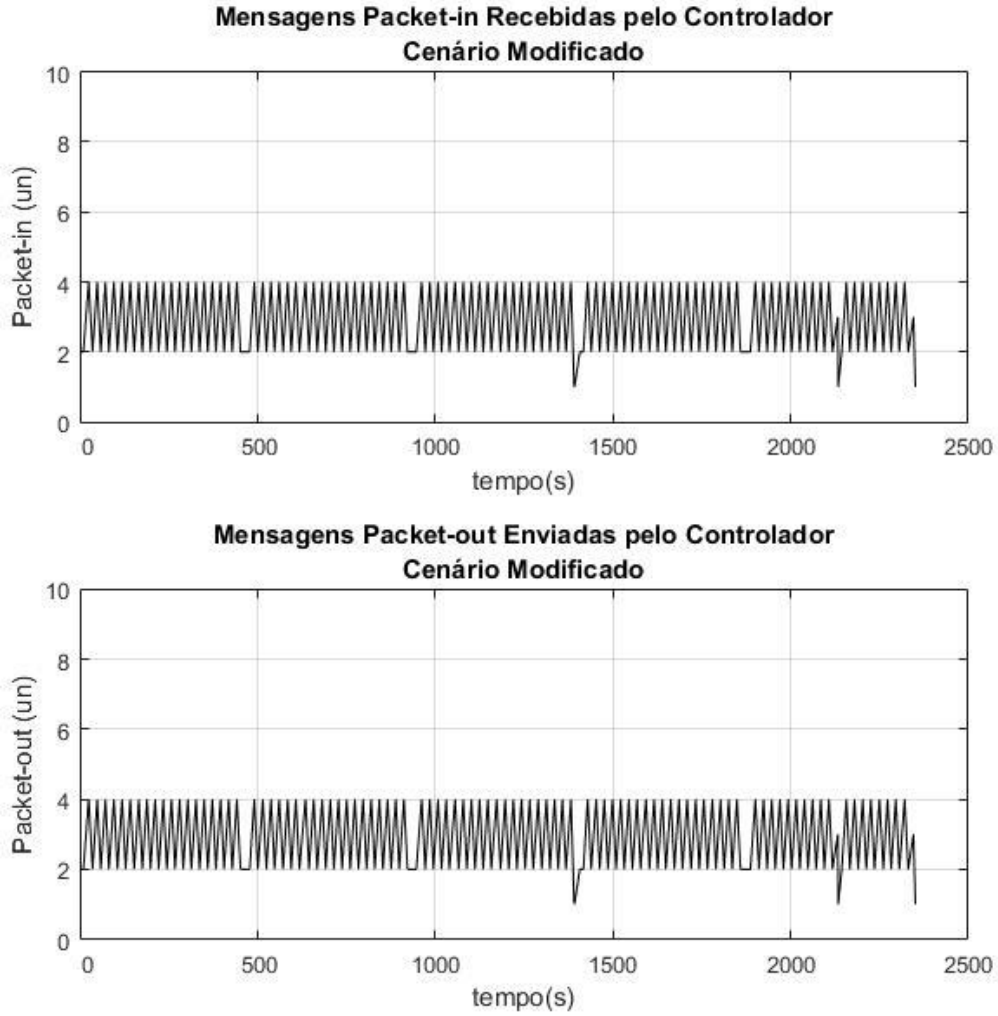
Figura 4.3 - Quantidade de *Packet-In* Recebidos e de *Packet-Out* Enviados pelo Controlador Durante a Execução dos Testes – Cenário I - Original



Fonte: elaborado pelo autor.

É possível verificar que o gráfico de envio de *packet-out* e de *packet-in* são semelhantes durante boa parte do tempo, mas, ao aumentar a quantidade de *packet-in* recebidos pelo controlador, o valor de *packet-out* se manteve no ritmo inicial. No cenário I, com o mecanismo RDP, o comportamento da rede se manteve constante, uma vez que o envio de *packet-in* é controlado para que não sejam enviadas mensagens sucessivas referente a um mesmo fluxo, como pode ser visto nos gráficos exibidos na figura Figura 4.4.

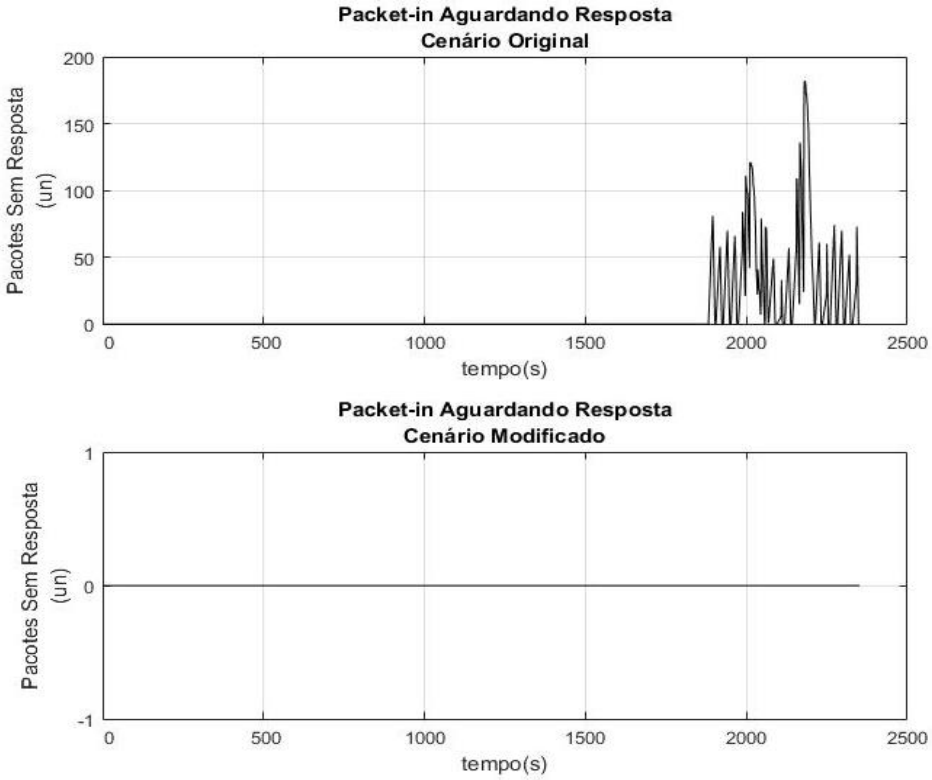
Figura 4.4 - Quantidade de *Packet-In* Recebidos e de *Packet-Out* Enviados pelo Controlador Durante a Execução dos Testes – Cenário I - Modificado



Fonte: elaborado pelo autor.

A diferença entre os comportamentos dos cenários fica evidente quando se analisa o tempo que um *packet-in* aguarda para ser respondido. Nos gráficos abaixo (Figura 4.5), é possível verificar quantas mensagens assíncronas aguardam uma resposta por mais de segundo em ambos os cenários (sem e com o mecanismo RDP).

Figura 4.5 - Comparativo de Quantidade de *Packet-In* sem Resposta a cada Segundo Durante o Teste



Fonte: elaborado pelo autor.

A diferença de resultado da métrica do tempo necessário para processar uma mensagem *packet-in* é compreendida quando analisada a taxa de processamento dessas mensagens. No momento que foram enviadas mais mensagens *packet-in* por segundo que o controlador consegue processar em um segundo, o tempo necessário para responder a essas mensagens tende a ser maior.

4.2.1.3 Mensagens Error e Warning

Utilizando o tráfego capturado, foi possível identificar anomalias ocorridas durante os testes. As anomalias descritas na Tabela 4.2 estão relacionadas à comunicação entre o cliente *OpenFlow* e o controlador.

Tabela 4.2 - Informações Gerais Obtidas do Tráfego Capturado Durante a Execução dos Testes

Tipo	Descrição	Cenário Modificado*	Cenário Original*
Error	Retransmission	0	12
Warning	TCP window specified by the receiver is now completely full	0	137
	TCP Zero Window segment	0	4584

*Valores retirados do Wireshark

Fonte: elaborado pelo autor.

Conforme a tabela anterior, é possível identificar que, no cenário original (sem a implementação do Mecanismo RDP), houve a ocorrência das anomalias do tipo

"*Error*" e "*Warning*". Enquanto no cenário modificado nenhuma anomalia desses dois tipos foi identificada.

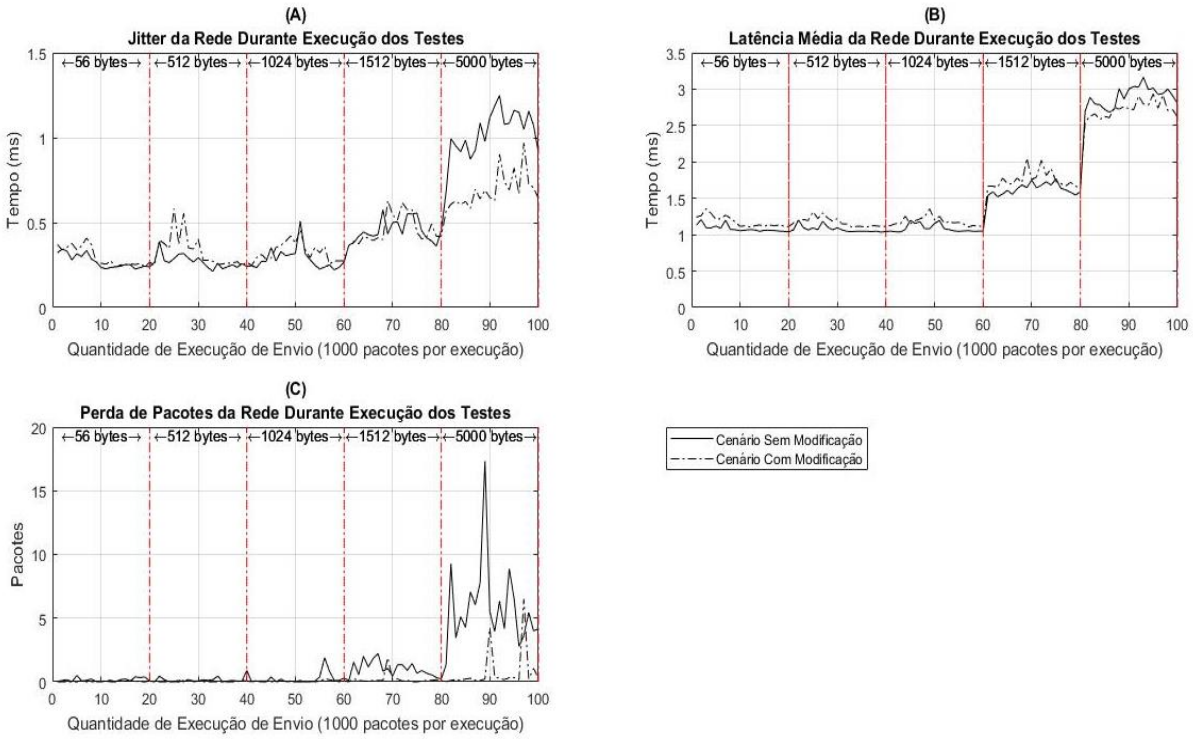
Com os resultados obtidos, é possível afirmar que, pelo fato de se ter uma grande incidência das anomalias ***"TCP window specified by the receiver is now completely full"*** e ***"TCP Zero Window"***, a anomalia de "***Retransmition"*** tenha ocorrido por um estouro de buffer no controlador. Já as anomalias do tipo "*Warning"* são provenientes da carga excessiva de pacotes e com tamanho próximos do MTU. Sendo assim, o emprego do mecanismo RDP preserva o buffer estabelecido no início da conexão entre cliente e controlador *OpenFlow*, evitando, assim, a ocorrência de estouro de buffer e de retransmissões.

4.2.1.4 "Saúde" da rede

Os dados referentes ao desempenho da rede também foram coletados durante a realização dos testes. Com esses dados, as seguintes informações foram obtidas: latência média da rede, *jitter* e os pacotes perdidos em cada teste.

O desempenho da rede se mantém em ambos cenários, tanto no cenário sem e com a modificação aqui proposta. É importante verificar que há uma pequena diferença no *jitter* (A) e na latência (B) da rede (Figura 4.6), em torno de 0.1 ms, entre os cenários. Contudo, a perda de pacotes (C) é maior no cenário sem a modificação (Figura 4.6).

Figura 4.6 - *Jitter* (A), Latência (B) e Perda de Pacotes (C) da Rede Durante a Execução dos Testes - Cenário I



Fonte: elaborado pelo autor.

A diferença nos valores da latência e *jitter* pode ser compreendida quando analisada a carga de processamento e o consumo de memória do OVS durante a execução dos testes. Os valores descritos na Tabela 4.3 são provenientes da execução dos scripts apresentados no tópico anterior.

Tabela 4.3 - Consumo de Recursos de Hardware do OVS -Cenário I

Descrição	**Consumo Médio de Processador (%)**		**Consumo Médio de Memória RAM (%)**	
	Cenário Modificado	**Cenário Original**	**Cenário Modificado**	**Cenário Original**
OVS	28,54	26,93	0,13	0,10

Fonte: elaborado pelo autor.

Ao iniciar o *Mininet*, um processo do OVS é também iniciado. Trata-se do **"*ovs-vswitchd*"**, responsável por todos os switches do cenário. Ou seja, todas as cargas de todos os switches serão centralizadas em um único processo, não sendo possível realizar a medição individual de cada switch. É importante ressaltar que o mecanismo RDP é realizado por cada switch, seja a retirada do *payload* do pacote ingressante, seja a implementação do mecanismo deduplicação.

4.2.1.5 Consumo de Hardware do controlador

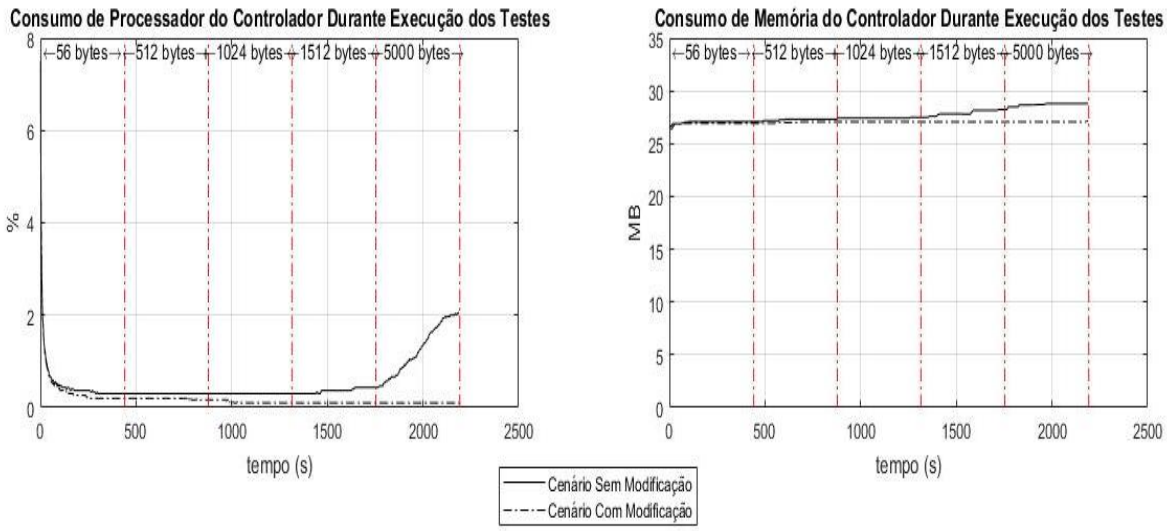
O principal objetivo desse trabalho é diminuir a carga no controlador, proveniente do envio de solicitações de entrada de fluxo. Por esse motivo, durante a execução dos testes, foram coletadas informações referentes ao consumo de processador, consumo de memória RAM e tráfego de entrada e saída na interface de rede.

As informações referentes ao consumo de processador e de memória foram coletadas da mesma maneira das do OVS, ou seja, armazenando as informações referentes ao processo da aplicação "*ryu-manager*", utilizada pelo controlador *Ryu*. O tráfego entrante, por sua vez, foi coletado via **tcpdump** e por meio do comando "**cat /proc/net/dev**".

É possível verificar que durante a execução dos testes, quando mecanismo RDP estava instaurado, o consumo do processador e de memória se manteve, o mesmo não ocorreu sem o mecanismo. Essa variação está diretamente ligada à quantidade de mensagens recebidas pelo Controlador. O gráfico abaixo (É possível verificar que, durante a execução dos testes, quando mecanismo RDP estava implementado, o consumo do processador e de memória se manteve, o mesmo não ocorreu sem o mecanismo. Essa variação está diretamente ligada à quantidade de mensagens recebidas pelo Controlador.

Figura 4.7) demonstra o comportamento do controlador no que diz respeito ao consumo de seu processador e de memória, durante a execução dos testes. É possível verificar que, durante a execução dos testes, quando mecanismo RDP estava implementado, o consumo do processador e de memória se manteve, o mesmo não ocorreu sem o mecanismo. Essa variação está diretamente ligada à quantidade de mensagens recebidas pelo Controlador.

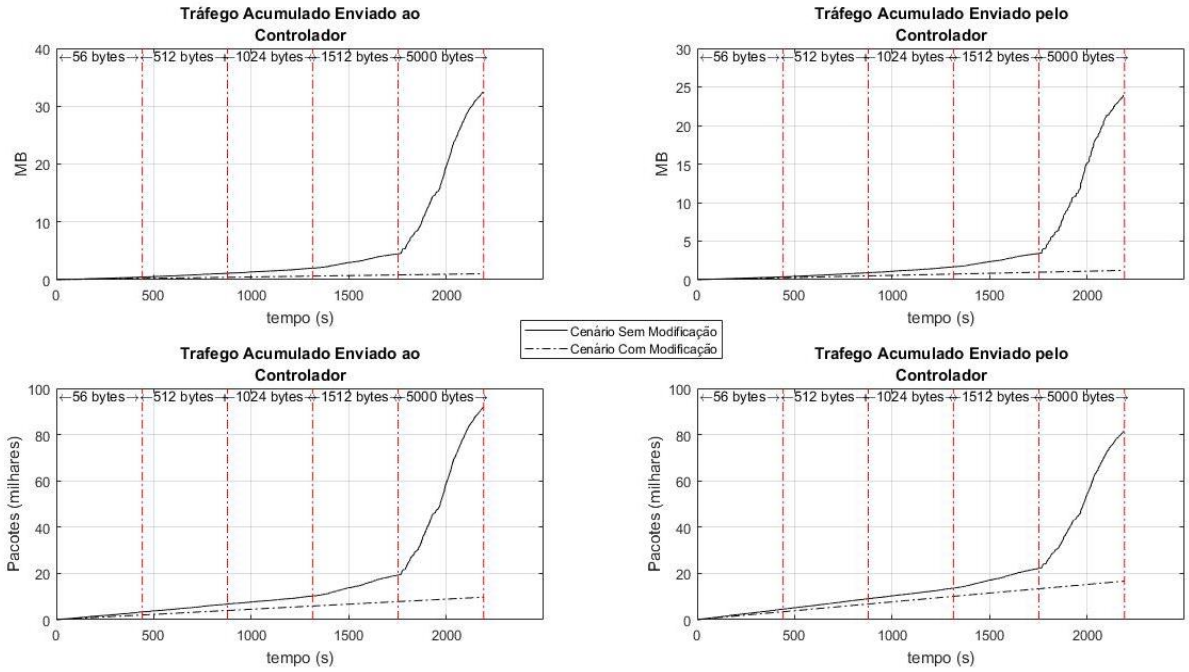
Figura 4.7 - Consumo de Processador e Memória do Controlador



Fonte: elaborado pelo autor.

Por fim, os valores de tráfego de entrada e saída do controlador foram armazenados de forma que fosse possível analisar o comportamento desses no emprego do mecanismo RDP. Referente a esse tráfego, as informações coletadas fazem referência à quantidade de bytes e pacotes de entrada e saída durante os testes. Esses gráficos podem ser vistos na Figura 4.8.

Figura 4.8 - Tráfego em Bytes e em Quantidade de Pacotes de Entrada e Saída do Controlador



Fonte: elaborado pelo autor.

4.2.2 Cenário II

Por se tratar de um cenário em um ambiente real, os equipamentos com o OVS instalado tiveram seus dados coletados no momento de utilização da infraestrutura pelos os usuários dos locais onde foram instalados os equipamentos. Nesse cenário, o roteiro de teste consistiu em reiniciar os serviços que estão sobre infraestrutura SDN. A partir desse momento, foram iniciadas as coletas dos dados desses serviços, do controlador e dos equipamentos com o OVS.

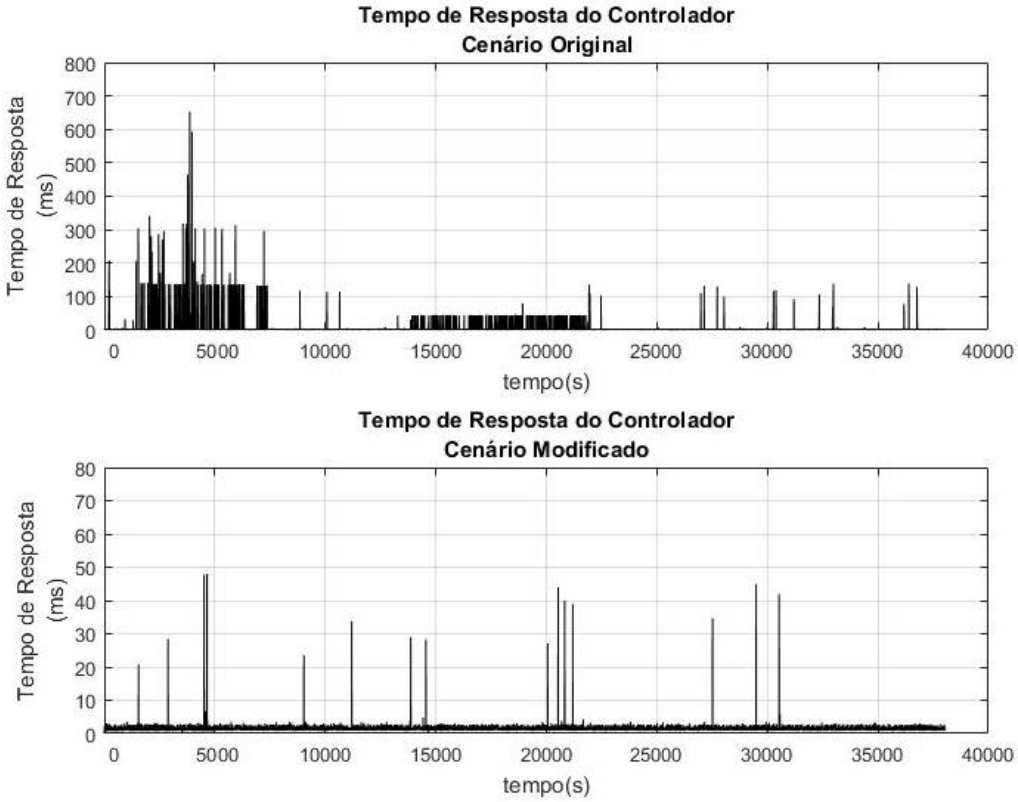
A avaliação do comportamento do controlador em um cenário de produção real foi feita sem a execução de testes de carga ou de “stress”. Isso possibilitou verificar se, em um ambiente real, o emprego do mecanismo traz benefícios para o controlador e se ele traz algum impacto à infraestrutura de rede e seus serviços.

4.2.2.1 Tempo necessário para o processamento de mensagens assíncronas

O cálculo utilizado (Equação 1) e a forma de captura do tráfego foram os mesmos do cenário I. O cálculo do tempo de resposta de um *packet-in* foi realizado apenas quando este possuía um *packet-out* correspondente. Foram consideradas correspondentes mensagens que tinham o número de referência no pacote com *packet-out* igual ao número de sequência do pacote com mensagem *packet-in*, bem como quando o pacote encapsulado era igual em ambas mensagens (*packet-in* e *packet-out*).

O **TPMA** obtido nesse cenário sem e com a estratégia proposta é 2ms e 1ms, respectivamente. Apesar de a diminuição ser aparentemente pequena, é importante verificar o tempo de resposta do controlador para cada *packet-in* recebido nesse cenário (sem e com modificação proposta) nos gráficos presentes na Figura 4.9.

Figura 4.9 - Tempo Necessário para o Controlador Processar uma Resposta



Fonte: elaborado pelo autor.

Contudo, como se trata de um cenário com links diferentes e máquinas com configurações diferentes, a Tabela 4.4 separa o tempo necessário para processar mensagens assíncronas, por parte do controlador, por local.

Tabela 4.4 - Tempo Necessário para Processar uma Mensagem Assíncrona por Parte do Controlador

Descrição	**Cenário Modificado**			**Cenário Original**		
	Tempo Máximo	**Tempo Mínimo**	**Tempo Médio**	**Tempo Máximo**	**Tempo Mínimo**	**Tempo Médio**
Ifes–Campus Vila Velha (Externo)	4,9ms	0,8ms	1,3ms	6,4ms	0,8ms	1,5ms
Ifes–Campus Vila Velha (Interno)	4,3ms	0,7ms	1,3ms	6,3ms	0,7ms	1,4ms
Ifes–Campus Vitória	4ms	0,7ms	1,3ms	5,1ms	0,9ms	1,5ms
Ifes–Campus Serra	6,7ms	0,5ms	1,3ms	652,9ms	0,8ms	2,4ms
Ifc–Campus Videira	48ms	0,8ms	1,7ms	593,1ms	0,9ms	10,9ms

Fonte: elaborado pelo autor.

Sobre os dados apresentados na tabela acima, é importante ressaltar alguns pontos: os links que interligam os equipamentos com o OVS da rede Ifes Campus Vila Velha (externo e interno) com o controlador são uma LAN; o link que interliga o equipamento no Ifes Campus Vitória com o controlador tem uma velocidade 1

Gbps[13] de download e upload; a conexão entre os equipamentos do Ifes Campus Serra e IFC Campus Videira necessitam de mais saltos para chegar ao controlador instalado no Ifes Campus Vila Velha.

A variação de valores dos tempos descritos na tabela pode ser justificada pelo tamanho do *packet-out* enviado pelo controlador. Isso porque o tamanho da mensagem *packet-out* está diretamente relacionado ao tamanho da mensagem *packet-in*. Ainda, no caso de mais de um *packet-out* correspondente a um mesmo *packet-in*, foi considerado o tempo de envio do último em ambos os ambientes.

4.2.2.2 Taxa de Processamento de Mensagens Assíncronas

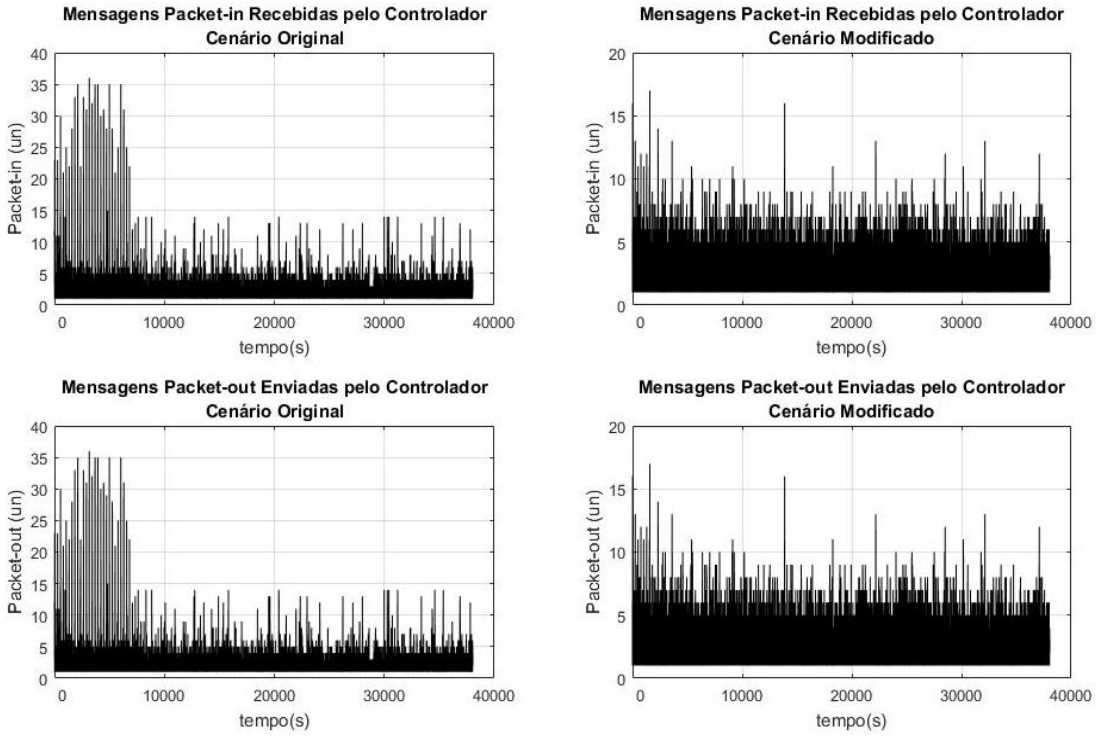
Como descrito no cenário I, essa métrica está relacionada à métrica anterior, onde um tempo maior para responder uma quantidade de mensagens ingressantes ocorre quando a taxa de processamento é menor que essa. Sendo assim, foi utilizada a mesma Equação 2 do cenário I para calcular a taxa de processamento, bem como o mesmo método de coleta de dados.

As mensagens assíncronas recebidas (*packet-in*) e as mensagens enviadas (*packet-out*) pelo controlador tiveram a mesma distribuição ao longo da coleta em cada ambiente, conforme os gráficos apresentados na

Figura 4.10.

[13] O Ifes Campus Vitória faz parte do NOC Metrovix, uma rede física composta por fibras ópticas que contornam o município de Vitória-ES - http://www.pop-es.rnp.br/?page_id=1021.

Figura 4.10 - Quantidade de *Packet-In* Recebidos e de *Packet-Out* Enviados pelo Controlador Durante a Execução dos Testes – Cenário II – Original e Modificado



Fonte: elaborado pelo autor.

Essa distribuição igual de entrada e saída de mensagens do controlador significa que todas eram respondidas tão logo fossem recebidas, não necessitando de mais de um segundo para serem processadas. Nesse cenário, a taxa de processamento máxima dos ambientes sem e com o mecanismo RDP foi de 36 e 17 mensagens por segundo, respectivamente.

Essa diferença na taxa de processamento em ambos os ambientes ocorre devido ao controle de envio simultâneo de mensagens assíncronas referentes a um mesmo fluxo presente no ambiente com o mecanismo RDP.

4.2.2.3 Mensagens Error e Warning

Ao analisar tráfego capturado, que era direcionado ao controlador, não foi possível identificar a ocorrência de retransmissões, do tipo ERROR, em ambos ambientes (com e sem o mecanismo). Não houve a ocorrência da anomalia "*TCP window specified by the receiver is now completely full*", do tipo ***Warning***, durante o período de captura em ambos ambientes. Dessa forma, a ocorrência ou não da anomalia "*TCP Zero Window Segment*", do tipo ***Warning***, pode ser ignorada. Sendo assim, não há dados relevantes a serem apresentados nessa métrica.

4.2.2.4 "Saúde" da rede

Os serviços que estão no cenário II foram monitorados para que fosse possível comparar o comportamento da infraestrutura com e sem o mecanismo RDP. Na Tabela 4.5, estão os valores de latência da rede para os serviços utilizados.

Tabela 4.5 - Desempenho dos Serviços de Redes - Cenário II

Serviço	**Descrição**	**Latência**	
		Cenário Modificado*	**Cenário Original***
Serviço - VOIP	Tronco IAX - Campus Vila Velha x Campus Serra	5ms	6ms
	Tronco IAX - Campus Vila Velha x Campus Videira	44ms	42ms
	Ramal SIP - Servidor Campus Vila Velha x Ramal 0817 Campus Vila Velha	2ms	2ms
Serviço de Impressão	Servidor LPD Windows - Ifes Campus Vila Velha	<1ms	<1ms
Servidores de Pesquisa	Servidores Linux	<1ms	<1ms

Fonte: elaborado pelo autor.

Nos serviços VoIP foram utilizados os valores indicados na aplicação Elastix, seja para as conexões IAX e SIP. Para a impressora, foi utilizado o valor de conexão com o servidor de impressão e a máquina cliente. Por fim, para mensurar o impacto na infraestrutura de redes no tráfego multimídia, entre os Campi Vila Velha e Vitória do Ifes, foi utilizada a ferramenta da solução de [35]. Os valores obtidos durante a execução dos testes na rede multimídia sobre a infraestrutura SDN implementados no cenário II podem ser vistos na Tabela 4.6.

Tabela 4.6 - Resultado do Desempenho da Rede Multimídia

Descrição	**Taxa de Chegada**		**Perda (%)**	
	Cenário Modificado	**Cenário Original**	**Cenário Modificado**	**Cenário Original**
Teste com 2000 chamadas	4	4	28,33	28,73
Teste com 4000 chamadas	4	4	29,71	28,15

Fonte: elaborado pelo autor.

Assim como no cenário I, não houve alterações significativas nos valores dos serviços que utilizaram a infraestrutura. Tal informação é importante, pois, assim como no cenário I, os equipamentos com OVS instalados, quando tiveram a

implementação do mecanismo RDP, tiveram um aumento no consumo de memória RAM em relação à implementação sem esse mecanismo, conforme Tabela 4.7.

Tabela 4.7 - Consumo de Recursos de Hardware dos Equipamentos - Cenário II

Descrição	**Consumo Médio de Processador (%)**		**Consumo Médio de Memória RAM (%)**	
	Cenário Modificado	**Cenário Original**	**Cenário Modificado**	**Cenário Original**
OVS-INT – Ifes Campus Vila Velha	0,33	0,02	0,34	0,01
OVS-VV - Ifes Campus Vila Velha	0,20	0,12	0,20	0,10
OVS-VIX - Ifes Campus Vitória	0,20	0,19	0,18	0,10
OVS-SR - Ifes Campus Serra	0,26	0,30	0,29	0,10
OVS-VID – Ifc Campus Videira	0,20	0,20	0,20	0,10

Fonte: elaborado pelo autor.

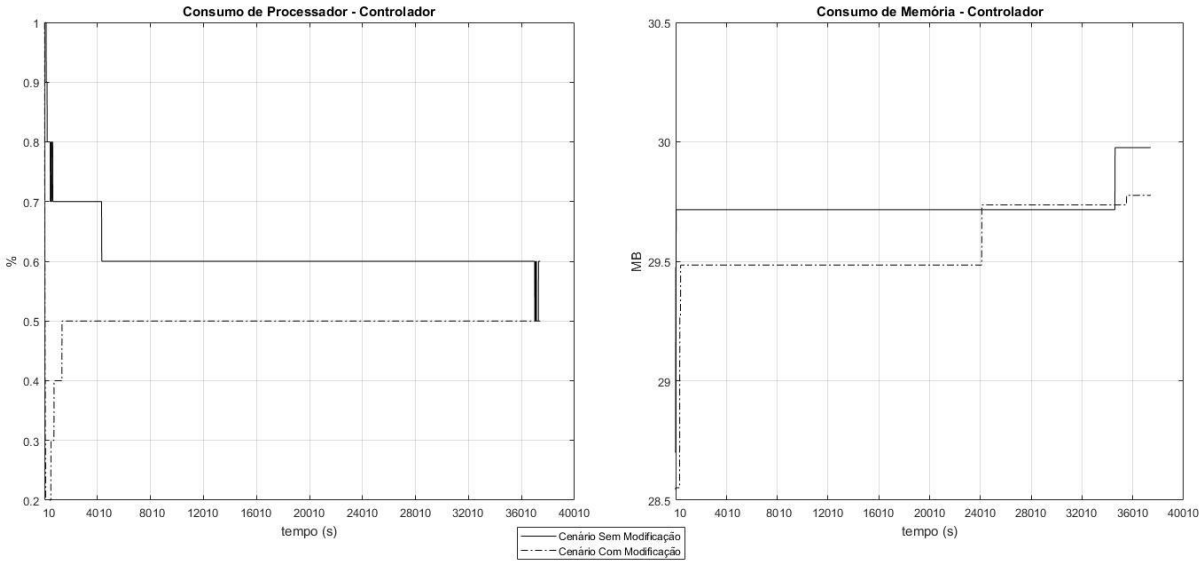
4.2.2.5 Consumo de Hardware do controlador

Durante o período em que foi capturado o tráfego do controlador (entrada e saída), as informações referentes ao consumo de processador, de memória RAM e tráfego de entrada e saída na interface de rede também foram coletadas. Foram coletadas as informações de consumo do processo criado a partir da aplicação "*ryu-manager*", utilizada pelo controlador *Ryu*. Já o tráfego entrante foi coletado via **tcpdump** e por meio do comando "**cat /proc/net/dev**".

No que diz respeito ao consumo de memória e processador do controlador, foi observada pouca variação nos dois ambientes, pois não foi realizado teste de envio de pacotes de diferentes tamanhos ou em grandes quantidades em pequenos intervalos, como no Cenário I. O consumo desses recursos do controlador, durante o período de utilização dos serviços que estão sobre a infraestrutura SDN, pode ser vistos nos gráficos ilustrados na .

Figura 4.11.

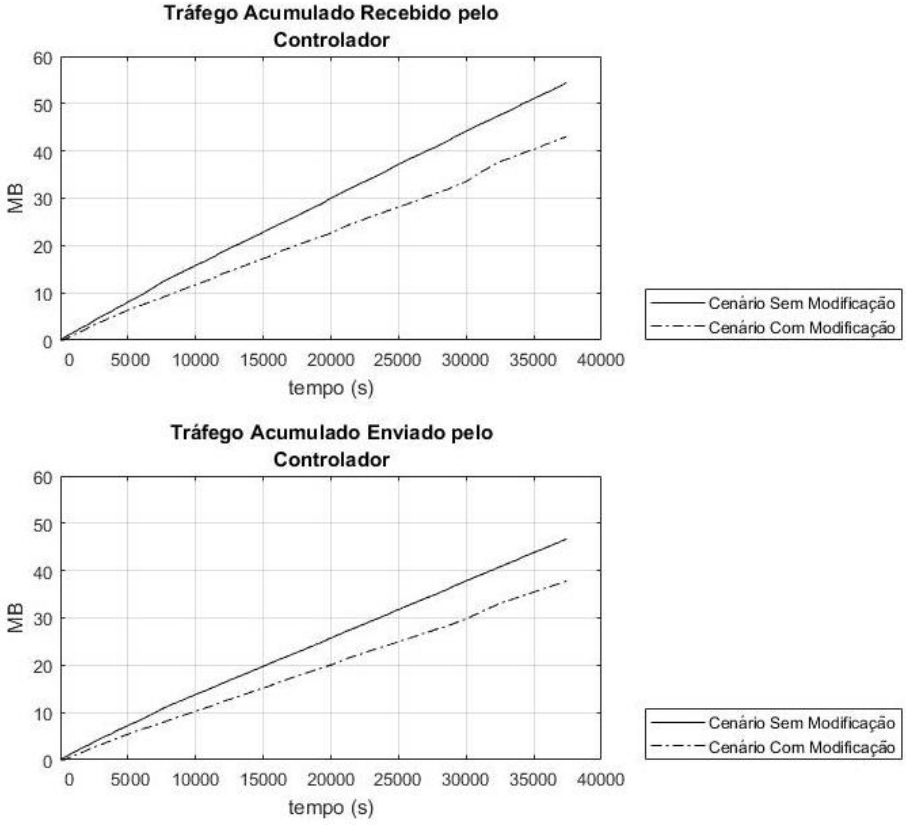
Figura 4.11 - Consumo de Processador e Memória do Controlador



Fonte: elaborado pelo autor.

É possível identificar uma pequena diminuição no consumo de memória e processamento do controlador quando os equipamentos estão com o mecanismo RDP implementado no OVS. Contudo, quando analisada a totalidade de tráfego de rede enviado e recebido pelo controlador, há uma considerável diminuição quando o ambiente está com o mecanismo RDP implementado (Figura 4.12).

Figura 4.12 - Tráfego de Rede do Controlador (MegaBytes Enviados e Recebidos)

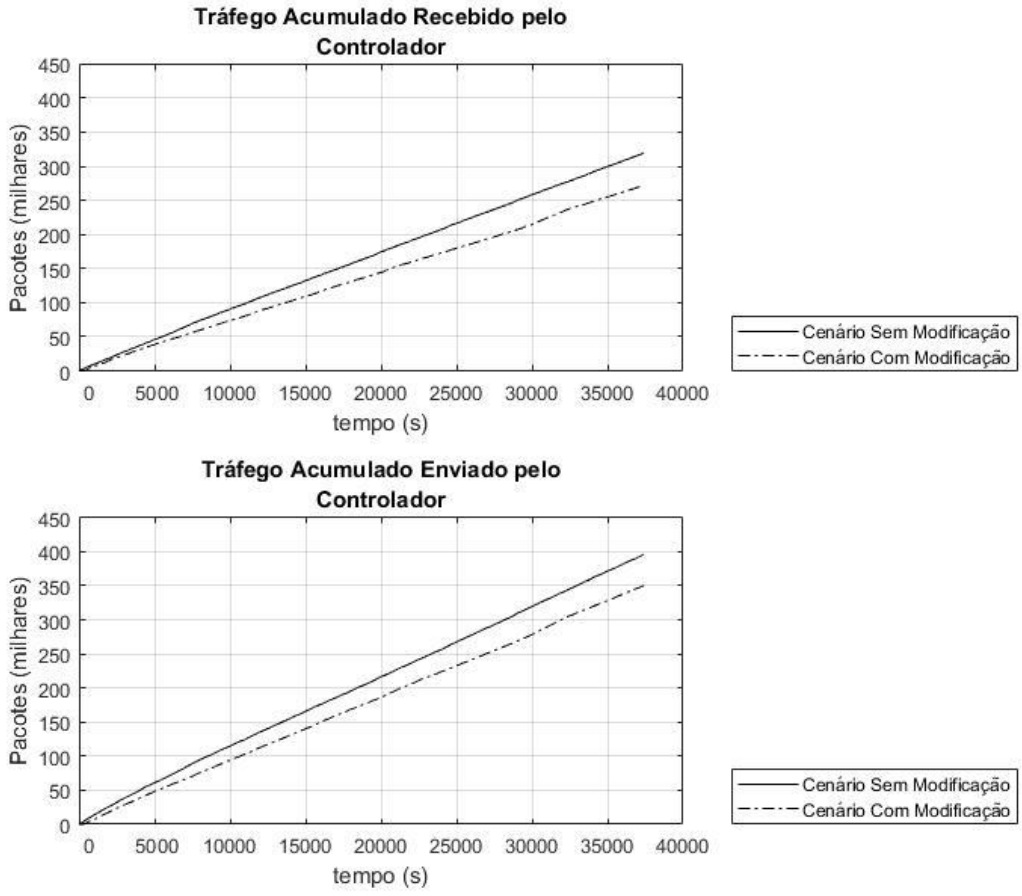


Fonte: elaborado pelo autor.

Nesse caso, como se trata de um cenário real, a diminuição da carga direcionada ao controlador também traz uma diminuição de tráfego ao link onde este foi implementado. E não apenas o link do controlador, mas também os links que interligam os equipamentos instalados.

Além da diminuição no tráfego enviado e recebido pelo controlador, foi possível identificar uma diminuição na quantidade de pacotes enviados ao controlador (Figura 4.13).

Figura 4.13 - Tráfego de Rede do Controlador (Pacotes Enviados e Recebidos)



Fonte: elaborado pelo autor.

Com isso, observou-se que os valores diminuíram em ambos os gráficos, com pacotes menores e evitando os pacotes duplicados. Conforme consta na Tabela 4.8, os pacotes enviados ao controlador, e que são mensagens *packet-in*, tiveram o tamanho máximo e a média maiores em ambientes sem o mecanismo RDP implementado.

Tabela 4.8 - Tamanho dos Pacotes - Cenário II

Descrição	Cenário Modificado*	Cenário Original*
Tamanho Máximo	168	1482
Tamanho Médio	162	191

Fonte: elaborado pelo autor.

4.2.3 Considerações sobre os Resultados Obtidos

Com os resultados de cada cenário, é possível verificar que o mesmo benefício de um cenário ocorreu no outro. Sendo assim, independente do mecanismo estar em uma infraestrutura emulada (cenário I) ou real (cenário II), houve diferença quando o ambiente possuía o mecanismo RDP implementado.

Dentre as diferenças encontradas, destacam-se:

- **i.** diminuição do tráfego de rede do controlador: tal resultado era esperado tendo em vista que o pacote a ser encapsulado na mensagem *packet-in* teve seu tamanho reduzido e foi evitado o envio de seguidas mensagens *packet-in* referente a um mesmo fluxo;
- **ii.** diminuição na carga de processamento e de memória RAM do controlador: pelo fato dos testes executados no cenário I possuírem pacotes maiores em quantidade e tamanhos, a diminuição foi maior, em relação ao cenário II, que teve uma pequena diminuição de consumo desses recursos, quando os switches estavam com o mecanismo RDP implementado;
- **iii.** menor tempo de resposta do controlador para mensagens assíncronas: em ambos os cenários, o tempo de resposta foi menor, pois os pacotes recebidos pelo controlador eram menores e não duplicados. Dessa forma, as regras são aplicadas em um tempo menor;
- **iv.** manutenção do desempenho da rede: os serviços que estavam sobre a infraestrutura dos cenários tiveram seu desempenho mantido. Portanto, ainda que o consumo de recursos (processador e memória RAM) dos equipamentos com os clientes *OpenFlow* tenha tido um aumento elevado quando comparado ao cenário sem a modificação. Porém não ultrapassou o 1% de ambos os recursos, não influenciando no desempenho da rede.

Sobre as diferenças apontadas, é importante ressaltar que, no cenário I, os clientes *OpenFlow* já possuíam regras pré-estabelecidas e, por isso, os dados eram enviados sem necessariamente aguardar a resposta do controlador. O cenário II, porém, foi planejado para proceder com os fluxos de comunicação apenas com as regras provenientes do controlador. Tratando-se de um cenário estabelecido em uma infraestrutura real e em produção, a diminuição no tempo de resposta influencia diretamente na liberação ou não do fluxo do pacote.

Ao evitar o envio de pacotes de um mesmo fluxo seguidos e diminuir o tamanho do pacote encapsulado, diminui-se também o tráfego total da rede. Isso é importante quando o link de comunicação do controlador com os clientes *OpenFlow* é compartilhado por outros serviços (ex.: acesso à internet).

5 CONCLUSÕES

Este trabalho apresentou uma estratégia para o estabelecimento de novos fluxos em redes SDN/*OpenFlow* com o objetivo de diminuir a carga direcionada ao controlador. A estratégia apresentada foi implementada em cenários emulados e não emulados, a partir da alteração do código fonte do OpenVSwitch. Essa alteração consistiu em redimensionar o pacote encapsulado nas mensagens *packet-in* e estabelecer um controle de envio dessas mensagens.

No cenário emulado, o comportamento do controlador foi avaliado, a partir do envio de mensagens ICMP entre os hosts. Essas mensagens tiveram seus tamanhos modificados para que fosse possível avaliar o impacto que estas causam no desempenho do controlador.

Sem o mecanismo RDP, foi possível chegar à conclusão de que quanto maior a quantidade de mensagens e seu tamanho, maior também é o tempo que o controlador necessita para processá-las. Não apenas o tempo é alterado, como também a quantidade de mensagens que o controlador consegue processar a cada segundo. Com o mecanismo RDP, o controlador teve um comportamento padrão durante todo o teste, sem que houvesse variações em seu tempo de resposta, sendo bem menor o tempo necessário para processar as mensagens *packet-in* em relação ao cenário tradicional. Isso levou a uma taxa de processamento igual à taxa de recebimento de mensagens *packet-in* no controlador. Conclui-se, portanto, que, ao encapsular apenas as informações referentes ao fluxo do pacote, a mensagem *packet-in* diminui de tamanho e, consequentemente, o tráfego direcionado ao controlador também diminui. Isso ocorreu sem que houvesse qualquer alteração na comunicação entre os hosts, ou seja, o funcionamento da infraestrutura permaneceu inalterado.

Além disso, foi possível concluir que a quantidade de mensagens *packet-in* enviadas pode ser maior que a necessária. Dois fatos apontam isso: pacotes fragmentados no cliente *OpenFlow* por serem maiores que o MTU configurado; e pacotes ingressantes de um mesmo fluxo recebidos pelo cliente *OpenFlow* antes que a resposta do controlador à primeira mensagem packet-in de fluxo seja recebida pelo cliente OpenFlow. Em ambos os casos, com o mecanismo RDP, o envio de mensagens duplicadas, referentes a um mesmo fluxo, não foi realizado, o que também diminuiu a carga direcionada no controlador.

O cenário operacional utilizou de máquinas físicas e virtuais, com os clientes *OpenFlow* que compunham a infraestrutura de rede SDN/*OpenFlow* – Ifes separados. Além disso, nesse cenário, o tráfego gerado é proveniente da execução de serviços em produção. Novamente, houve uma melhora no desempenho do controlador quando o cliente *OpenFlow* possuía o mecanismo RDP instalado. Em ambos, a taxa de processamento de mensagens *packet-in* foi igual à taxa de chegada dessas mensagens. Porém, mesmo que o controlador conseguisse processar as mensagens em menos de um segundo, o tempo necessário para processar essas mensagens foi maior quando os clientes *OpenFlow* não possuíam o mecanismo RDP. Quando este estava implementado, o tempo de resposta se manteve dentro de um padrão semelhante ao cenário emulado.

Conclui-se, portanto, que o controlador teve uma carga menor quando o mecanismo RDP estava implementado nos clientes *OpenFlow* em ambos os cenários. Os fatores que contribuíram para essa redução foram o tamanho padronizado dos pacotes a serem encapsulados nas mensagens *packet-in* e o envio da quantidade necessária dessas mensagens. Não apenas no que diz respeito ao tráfego de rede, mas também o consumo de memória RAM e processador do controlador.

5.1 Contribuições do Trabalho

Com os resultados obtidos nesse trabalho, é possível agora adotar um mecanismo que aperfeiçoe as mensagens *packet-in* de forma que o controlador receba apenas as informações necessárias sobre o fluxo a ser consultado. E, ao mesmo tempo, deve-se considerar que a taxa de processamento de mensagens assíncronas por parte do controlador impacta diretamente no tempo que ele necessita para processá-las e que o envio de mais de uma mensagem *packet-in*, de um mesmo fluxo, pode acarretar em uma taxa de recebimento maior que a taxa de processamento, o que leva a um tempo maior para processar tais mensagens.

Outro ganho obtido com a adoção dessa estratégia está em uma diminuição do tráfego no link que interliga o controlador aos clientes *OpenFlow*. Em redes emuladas, é fato que os links estão em um barramento sem concorrer com outros serviços e, por isso, nenhum benefício do ponto de vista prático é verificado. Contudo, ao se implementar esse mecanismo em uma infraestrutura operacional, diminuir o tráfego no link utilizado para comunicação entre o controlador e os

clientes *OpenFlow*, sem prejudicar o seu desempenho, traz benefícios não apenas ao controlador como a toda infraestrutura de rede, uma vez que o link será poupado para atender às demandas de outros tráfegos existentes.

Ao combinar a proposta deste trabalho com a arquitetura proposta em [16], obtém-se uma infraestrutura ainda mais robusta. Isso porque não há necessidade de guardar em memória nenhum pacote completo, mas sim apenas o cabeçalho do pacote ingressante. Isso se deve ao fato da arquitetura descrita em [16] os sucessivos pacotes serem encaminhados pela ação das regras pré-estabelecidas.

Somado a esses benefícios, como o cenário implementado no Ifes necessitava de requisições sendo enviadas e recebidas por diversos serviços, foi utilizada a infraestrutura desse cenário para atender a duas demandas do Ifes Campus Vila Velha: disponibilizar acesso externo para o servidor VoIP para possibilitar a adoção do FoneRNP e possibilitar o acesso às máquinas de pesquisa de química para os professores na infraestrutura de rede. Em ambos os casos, não havia disponibilidade de portas nos switches e firewall.

5.2 Dificuldades e Limitações

Salienta-se que, durante o desenvolvimento da proposta deste trabalho e sua implementação, algumas dificuldades e limitações foram identificadas. Ao identificá-las, foi possível compreendê-las e buscar uma solução, de modo que a viabilidade do trabalho não fosse comprometida.

Propor uma nova estratégia no estabelecimento de novos fluxos necessita alterar a forma como os clientes *OpenFlow* compõem suas mensagens *packet-in*. Essa alteração pode ser realizada apenas em clientes *OpenFlow*, cujo código do software pode ser manipulado. Diante disso, este trabalho limitou-se a ser implementado e testado em clientes *OpenFlow* provenientes do software OpenVSwitch. Tal limitação só não acarretou na inviabilidade do referido trabalho, pelo fato desse software ser utilizado no Mininet e em equipamentos que utilizem o sistema operacional Linux.

Além de ter acesso ao código fonte do software, é importante que esse código esteja bem estruturado e documentado. Os comentários inseridos no código são essenciais para uma boa documentação, sendo uma peça importante para o entendimento do código fonte com relação ao seu desenvolvimento e manutenção

[36]. Por esse motivo, a ausência dos comentários na estrutura dos arquivos que compõem o OpenVSwitch tornou difícil a identificação das funções e estruturas utilizadas na composição de uma mensagem *packet-in* e na decodificação de uma mensagem *packet-out*.

Feita a alteração no código fonte para que os clientes *OpenFlow* utilizassem a estratégia aqui proposta, foi necessário estabelecer os cenários de teste e validação. No cenário emulado, a principal dificuldade encontrada foi o envio de pacotes MPLS próximo ao MTU definido. Isso porque, ao gerar um pacote de tamanho igual o MTU definido, os campos utilizados em pacotes MPLS por algum motivo são desconsiderados no cálculo do tamanho máximo do pacote. Sendo assim, as mensagens não eram enviadas. Essa dificuldade foi resolvida após configurar em cada instância de cliente *OpenFlow* criada no Mininet o tamanho máximo de MTU que suportasse a inserção do cabeçalho MPLS.

No cenário operacional ou cenário Ifes, houve a limitação no número de interfaces físicas disponível em cada equipamento. Além desse aspecto, houve também uma limitação na quantidade de campi que dispuseram de estações para compor o cenário.

A tabulação dos dados coletados durante a execução dos testes nos cenários foi a maior dificuldade encontrada nesse trabalho. Isso porque foi necessário utilizar vários campos para correlacionar as mensagens *packet-in* e *packet-out*. Por esse motivo, mensagens *packet-in* que não possuíam *packet-out* correspondente em cem por cento dos campos não foram consideradas nas métricas. Tal dificuldade, porém, não foi encontrada no ambiente com o mecanismo RDP.

5.3 Trabalhos Futuros

Ao analisar o comportamento do controlador no processamento de mensagens assíncronas, alguns pontos passíveis de aprimoramento podem ser objetos de novas pesquisas. O monitoramento do tempo necessário para processar as mensagens recebidas pelo controlador pode ser um fator interessante a ser abordado, pois, tendo essa métrica em tempo real, mudanças podem ser adotadas no controlador ou nos clientes *OpenFlow* visando diminuir esse tempo.

A associação das mensagens *packet-in* e *packet-out* também é um ponto a ser abordado posteriormente, pois, ao receber várias mensagens *packet-in*, o

controlador identifica as mensagens *packet-out*, a partir da última mensagem *packet-in* recebida. Dificulta-se, por conseguinte, o cálculo das métricas citadas nesse trabalho.

Ainda que o impacto não tenha influenciado no desempenho da rede, o emprego do mecanismo RDP acarretou em um aumento no consumo de memória dos equipamentos. É possível que o *buffer* de *packet-in* implementado seja desenvolvido conforme o *buffer* do OVS, o que diminuiria o consumo de *hardware* do equipamento.

REFERÊNCIAS

[1] Hu, F. Hao, Q. Bao, K. “A Survey on Software-Defined Network and *OpenFlow*: From Concept to Implementation,” *IEEE COMMUNICATION SURVEYS & TUTORIALS,* pp. 2181 - 2206, 2014.

[2] Nunes, B. Mendonca, M. Nguyen, X.-N. Obraczka, K. Turletti, T. “A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks,” *IEEE Communications Surveys & Tutorials,* pp. 1617 - 1634, 2014.

[3] McKeown, N. Anderson, T. Balakrishnan, H. Parulkar, G. Peterson, L. Rexford, J. Shenker, S. e Turner, J. “*OpenFlow*: Enabling Innovation in Campus Networks,” *ACM SIGCOMM Computer Communication Review,* pp. 69-74, 2008.

[4] ONF (Open Networking Foundation). Disponível em: http://www.opennetworking.org/. Acesso em agosto 2016.

[5] ONF (Open Networking Foundation). “*OpenFlow* Switch Specification”. Disponível em: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.3.5.pdf. Acesso em agosto 2016.

[6] Lara, A. Kolasani, A. Ramamurthy, B. “Network Innovation using *OpenFlow*: A Survey,” *IEEE Communications Surveys & Tutorials,* pp. 493 - 512, 2014.

[7] Cisco. “Cisco Application Centric Infrastructure”. Disponível em: http://www.cisco.com/c/en/us/solutions/data-center-virtualization/application-centric-infrastructure/index.html#~benefits,. Acesso em agosto 2016

[8] Juniper. “Contrail Networking”. Disponível em: http://www.juniper.net/us/en/products-services/sdn/contrail/contrail-networking/. Acesso em agosto 2016.

[9] Dixit, A. Hao, F. Mukherjee, S. Lakshman, T. V. Kompella, R.R. ElastiCon: An Elastic Distributed SDN Controller. Symposium On Architecture For Networking And Communications Systems. p. 17-28. 2014.

[10] Erickson, D. “The beacon openflow controller”. Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking. pp. 13-18. 2013

[11] Shin, S. Yegneswaran, V. Porras, P. Gu, G. “AVANT-GUARD: Scalable and Vigilant Switch Flow Management in Software-Defined Networks”. ACM SIGSAC Conference on Computer & Communications Security. pp. 413-424. 2013.

[12] Yu, M. Rexford, J. Freedman, M. J. Wang, J. “Scalable Flow-Based Networking with DIFANE,” *ACM SIGCOMM’10,* pp. 351 - 362, 2010.

[13] Curtis, A. R. Mogul, J. C. Tourrilhes, J. Yalagandul, P. Sharma, P. Banerjee, S. “DevoFlow: Scaling Flow Management for High-Performance Networks,” *ACM SIGCOMM’11,* pp. 254 - 265, 2011.

[14] Wang, R. Butnariu, D. Rexford, J. “Openflow-based server load balancing gone wild”. Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services, Hot-ICE’11. pp.12-12. 2011.

[15] Zhang, Y. “An adaptive flow counting method for anomaly detection in SDN”. Proceedings of the ninth ACM conference on Emerging networking experiments and technologies. pp. 25-30 2013.

[16] Rodriguez, F. L. Campelo, D. R. “A robust SDN network architecture for service providers,” *IEEE Global Communications Conference (GLOBECOM),* pp. 8 -12. 2014.

[17] Tootoonchian, A. Ganjali, Y. “HyperFlow: A Distributed Control Plane for *OpenFlow*,” *InmWren10,* pp. 1-6, 2010.

[18] Galluccio, L. Milardo, S. Morabito, G. SDN-WISE: Design, prototyping and experimentation of a stateful SDN solution for WIreless SEnsor networks. IEEE Conference on Computer Communications (INFOCOM). pp. 513 – 521. 2015.

[19] Zimmermann, H. “OSI Reference Model--The ISO Model of Architecture for Open Systems Interconnection”. IEEE Transactions on Communications. p. 425 – 432. 1980

[20] A TCP/IP Tutorial. Disponível em: https://tools.ietf.org/html/rfc1180. Acesso em novembro de 2016.

[21] INTERNET CONTROL MESSAGE PROTOCOL (ICMP) – RFC 792. Disponível em: https://tools.ietf.org/html/rfc792. Acesso em novembro de 2016

[22] TRANSMISSION CONTROL PROTOCOL (TCP) – RFC 793. Disponível em: https://tools.ietf.org/html/rfc793. Acesso em novembro de 2016

[23] User Datagram Protocol (UDP) - RFC 768. Disponível em: https://tools.ietf.org/html/rfc768. Acesso em novembro de 2016.

[24] Pfaff, B. Pettit, J. Koponen, T. Jackson, E. J. Zhou, A. Rajahalme, J. Gross, J. Wang, A. Stringer, J. Shelar, P. Amidon, K. Casado, M. “The Design and Implementation of Open vSwitch”, NSDI 15, pp. 117 – 130, 2015.

[25] Shalimov, A. Zuikov, D. Zimarina, D. Pashkov, V. e Smeliansky, R. “Advanced Study of SDN/*OpenFlow* controllers,” *CEE-SECR '13 Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia,* 2013.

[26] Khondoker, R. Zaalouk, A. Marx, R. Bayarou, K. “Feature-based comparison and selection of Software Defined Networking (SDN) controllers,” *WCCAIS - World Congress on Computer Applications and Information Systems (WCCAIS),* 2014.

[27] R. P. Team, RYU SDN Framework - Using *OpenFlow* 1.3, Ryu Project Team. Disponível em: http://osrg.github.io/ryu/resources.html#books. Acesso em agosto de 2016

[28] Lantz, B. Heller, B. McKeown, N. “A Network in a Laptop: Rapid Prototyping for Software-Defined Networks,” *Hotnets-IX Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks,* 2010.

[29] S.-Y. Wang, “Comparison of SDN *OpenFlow* Network Simulator and Emulators: EstiNet vs. Mininet,” *IEEE Symposium on Computers and Communication (ISCC),* 2014.

[30] Perešíni, P. Kúzniar, M, Vasic´, N. Canini, M. Kostic´, D. OF.CPP: Consistent Packet Processing for OpenFlow. Proceedings of the second

ACM SIGCOMM workshop on Hot topics in software defined networking. p. 97-102. 2013.

[31] Terminology for Benchmarking SDN Controller Performance draft-bhuvan-bmwg-sdn-controller-benchmark-term-01. Disponível em: https://tools.ietf.org/html/draft-bhuvan-bmwg-sdn-controller-benchmark-term-01. Acesso em fevereiro de 2017.

[32] Benchmarking Methodology for SDN Controller Performance draft-ietf-bmwg-sdn-controller-benchmark-meth-03. Disponível em: https://tools.ietf.org/html/draft-ietf-bmwg-sdn-controller-benchmark-meth-03. Acesso em fevereiro de 2017.

[33] Pedroso, M. P., Ferreira, E. C., Hantao, L. W., Bogusz, S., Augusto, F. "*Identification of volatiles from pineapple (Ananas comosus L.) pulp by comprehensive two-dimensional gas chromatography and gas chromatography/mass spectrometry*". Journal of Separation Science-JCR. v. 34, p. 1547-1554. 2011.

[34] Filgueiras, P. R.; Domingos, E.; Silva, S. R. C.; Castro, E. V. R.; Ferreira, E. C.; Araujo, B. R. F.; Sena, S. S.; Pinheiro, L. U.; Pinto, F. E.; Aquino, L. F. M.; Romao, W. "*MicroNIR applied to Quality Control of Fuel ? part 3: Determination of total sulphur in diesel using portable instrument*". PETROPHASE 2016: the 17th International Conference on Petroleum Phase Behavior & Fouling, 2016, Helsingør. Lyngby: DTU Chemical Engineering / CERE. p. 163. 2016

[35] Santos, C. A. S., Saleme, E. B., Andrade, J. C. S. . A Systematic Review of Data Exchange Formats in Advanced Interaction Environments. International Journal of Multimedia and Ubiquitous Engineering, v. 10, p. 123-140. 2015.

[36] Steidl, D. Hummel, B. Juergens, E. "Quality analysis of source code comments". IEEE International Conference on Program Comprehension (ICPC). pp. 83-92. 2013.

[37] Pettit, J. Gross, J. Pfaff, B. Casado, M. "Virtual Switching in an Era of Advanced Edges". 2nd Workshop on Data Center - Converged and Virtual Ethernet Switching. ITC 22. 2010.