UNIVERSIDADE DO ESTADO DE SANTA CATARINA – UDESC
CENTRO DE CIÊNCIAS TECNOLÓGICAS – CCT
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO – BCC

DEVAIR DENER DAROLT

ESTUDO E COMPARAÇÃO DE ALGORITMOS PARA ENCAMINHAMENTO DE FLUXOS EM REDES DEFINIDAS POR SOFTWARE

JOINVILLE
2022

DEVAIR DENER DAROLT

ESTUDO E COMPARAÇÃO DE ALGORITMOS PARA ENCAMINHAMENTO DE FLUXOS EM REDES DEFINIDAS POR SOFTWARE

Monografia apresentada a disciplina de Trabalho de Conclusão de Curso II, do curso de ciência da computação do CCT/UDESC, como requisito para obtenção do título de bacharel em ciência da computação.

Orientador: Guilherme Koslovski

JOINVILLE
2022

DEVAIR DENER DAROLT

ESTUDO E COMPARAÇÃO DE ALGORITMOS PARA ENCAMINHAMENTO DE FLUXOS EM REDES DEFINIDAS POR SOFTWARE

Monografia apresentada a disciplina de Trabalho de Conclusão de Curso II, do curso de ciência da computação do CCT/UDESC, como requisito para obtenção do título de bacharel em ciência da computação.

Orientador: Guilherme Koslovski

BANCA EXAMINADORA:

Guilherme Piêgas Koslovski - Doutor (Orientador)
CCT/UDESC

Charles Christian Miers - Doutor
CCT/UDESC

Maurício Aronne Pillon - Doutor
CCT/UDESC

Joinville, 10 de Agosto de 2022

Aos meus pais, amigos e professores da Universidade do Estado de Santa Catarina, pela inspiração de sempre!

AGRADECIMENTOS

Agradeço ao meu orientador Guilherme P. Koslovski por aceitar dar continuidade ao meu trabalho que já estava um tempo no fundo da gaveta.

A todos os meus professores do curso de Ciência da Computação da Universidade do Estado de Santa Catarina – Udesc pela qualidade técnica e esforço de cada um em transmitir seus conhecimentos, incentivando a buscar melhores resultados sempre.

Aos meus pais que estiveram sempre ao lado apoiando ao longo de toda a minha vida. Sou grato a minha família por todo esse apoio, não teria conseguido sem.

Deixo um agradecimento especial ao meu orientador pela dedicação do seu tempo a todos os meus projetos de pesquisas, nos quais pude aprender diversos tópicos interessantes, foram tempos memoráveis. Quero agradecer a Deus por sempre ajudar a superar os desafios, evoluir e buscar a verdade em todas as situações.

RESUMO

O paradigma de Redes Definidas por Software (SDN) foi recentemente difundido como uma das possibilidades para sobrepor a inflexibilidade dos recursos de comunicação, sobretudo da Internet. Este paradigma, iniciado como um experimento acadêmico, vem sendo utilizado em cenários industriais (*e.g.,* provedores de nuvens computacionais, provedores de conteúdo) pela facilidade de gerenciamento e implantação de inovações. A verticalização existente em redes convencionais limitou o desenvolvimento de protocolos e propostas arquiteturais durante décadas. Em suma, SDN ultrapassou essa limitação através da separação dos planos de controle e dados nos dispositivos de comunicação. O gerenciamento do plano de controle fica a cargo de componentes conectados à infraestrutura (controladores) que atuam sobre os dispositivos de encaminhamento através de abstrações. Esse cenário flexível, permite a definição de novas políticas de controle e encaminhamento de fluxos por parte do administrador, implementadas diretamente no controlador. O alvo do presente trabalho é realizar um estudo comparativo entre políticas de encaminhamento de fluxos em SDN. Entre diversas estratégias existentes foram selecionadas 4 das mais comuns utilizadas. A primeira é a política de *round-robin* (RR), que usa a estratégia de fila circular para alternar entre os múltiplos caminhos existentes. A segunda política é a do Caminho Mais Curto Reativo (CMCR), na qual, o controlador escolhe a cada salto o próximo comutador a ser enviado, de modo a utilizar o caminho de menor quantidade de saltos. A terceira política utilizada foi o Caminho Mais Curto Proativo (CMCP) que implementa a lógica de menor quantidade de saltos, porém configurando todos os comutadores de uma única vez. Por último, a política de Caminho de Menor Tráfego (CMT) permite o controlador utiliza as informações de portas dos comutadores para encontrar o caminho com menor largura de banda utilizada. O cenário SDN será realizado com o emulador *Mininet* enquanto os algoritmos de encaminhamento serão implementados no controlador *Floodlight* e estudados com análise do desempenho da ferramenta *Numerical Aerodynamic Simulation* (NAS).

Palavras-chave: Engenharia de Tráfego, Floodlight, OpenFlow, Mininet, SDN.

ABSTRACT

The Software Defined Networks (SDN) paradigm has recently been widespread as one of the possibilities to overcome the inflexibility of communication resources, especially the Internet. This paradigm, started as an academic experiment, has been used in industrial scenarios (*e.g.*, cloud computing providers, content providers) due to the ease of managing and implementing innovations. The verticalization that exists in conventional networks has limited the development of protocols and architectural proposals for decades. SDN has overcome this limitation by separating control and data planes in communication devices. The control plane is managed by components connected to the infrastructure (controllers) that act on the forwarding devices through abstractions. This flexible scenario allows the definition of new control and flow forwarding policies by the administrator, implemented directly in the controller. The aim of this work is to carry out a comparative study between flow forwarding policies in SDN. Among several existing strategies, 4 of the most commonly used were selected. The first is the Round-Robin policy, which uses the circular queuing strategy to switch between existing multiple paths. The second policy is the shortest reactive path, in which the controller chooses at each hop the next switch to be sent, in order to use the path with the least number of hops. The third policy used was the proactive shortest path, in which the controller selects the path with the least amount of hops, but configuring all switches that belong to the path. Finally, the least traffic policy, which allows the controller to use the switch port information to find the path with the least bandwidth used. The SDN scenario will be performed with the Mininet emulator and the forwarding algorithms will be implemented in the Floodlight controller and analyzed through the performance of the Numerical Aerodynamic Simulation (NAS) tools.

Keywords: Floodlight, OpenFlow, Mininet, SDN, Traffic Engineering.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACL	*Access Control List*
API	Interface de Programação de Aplicações
ARP	*Address Resolution Protocol*
ASIC	*Application-Specific Integrated Circuits*
BT	*Block-Tridiagonal*
CG	*Conjugate gradient*
CMCP	Caminho Mais Curto Proativo
CMCR	Caminho Mais Curto Reativo
CMT	Caminho de Menor Tráfego
CPU	Unidade Central de Processamento
DRAM	Memória de Acesso Randômico Dinâmica
EP	*Embarrassingly parallel*
FIFO	*First In, First Out*
IIoT	*Industrial Internet of Things*
IP	*Internet Protocol*
IS	*Integer Sort*
LU	*Lower Upper*
MAC	*Media Access Control*
MG	*Multigrid*
MPI	*Message Passing Interface*
NAS	*Numerical Aerodynamic Simulation*
NASA	*National Aeronautics and Space Administration*
NFV	*Network Functions Virtualization*
NOS	*Network Operation System*
NTT	*Nippon Telegraph and Telephone*
OFRHM	*OpenFlow Random Host Mutation*
ONF	*Open Network Foundation*
QoS	*Quality of Service*
REST	*REpresentational State Transfer*
RR	*Round-Robin*

SDN	*Software-Defined Networking*
SLA	*Service Level Agreement*
SP	*Solution Pentadiagonal*
SSH	*Secure Shell Protocol*
SSL	*Secure Socket Layer*
TCAM	*Ternary Content- Addressable Memory*
TCP	*Transmission Control Protocol*
TLS	*Transport Layer Security*
URL	*Uniform Resource Identifier*
VLAN	*Virtual Local Area Network*

SUMÁRIO

1 INTRODUÇÃO

O gerenciamento de redes convencionais baseadas no *Internet Protocol* (IP) é um desafio para os administradores de redes. Para configurar diretrizes de segurança e encaminhamento de pacotes, é necessário utilizar instruções de baixo nível especificadas pelos fornecedores dos dispositivos. Tradicionalmente, cada dispositivo de rede possui uma camada de dados e uma camada de controle implementadas como parte do equipamento, tornando essas infraestruturas rígidas, com pouca possibilidade de inovação (KIM; FEAMSTER, 2013). Redes com alta complexidade de formulação e manutenção geralmente necessitam de intervenções manuais significativas em sua gestão, estando mais propensas a falhas e por isso são mais difíceis de serem atualizadas e gerenciadas (BENSON; AKELLA; MALTZ, 2009).

O conceito de Redes Definidas por Software (*Software-Defined Networking* – SDN), primeiramente desenvolvido como um experimento acadêmico, tornou-se uma tecnologia emergente para evolução das redes de computadores. Em suma, SDN introduziu a separação entre o plano de controle e o plano de dados, transformando os componentes de comunicação em simples dispositivos de encaminhamento. Neste cenário, a configuração das políticas de encaminhamento e balanceamento de carga pode ser implementada sobre um controlador logicamente centralizado, denominado *Network Operation System* (NOS) (KREUTZ et al., 2015).

A separação dos planos nos dispositivos de encaminhamento oferece uma série de benefícios para pesquisadores e administradores: *(i)* Ambiente simplificado e menos propenso a erros ocasionados por modificações em políticas. Ainda, as ações de configuração são realizadas por interfaces pré-definidas que abstraem os recursos reais. *(ii)* Algoritmos de controle podem automaticamente reagir a mudanças no estado da rede e manter as políticas de alto nível intactas. Assim, alterações podem ser realizadas nas tabelas de fluxo dos dispositivos de encaminhamento sem impactar a aplicação final. *(iii)* A centralização da lógica de controle em um controlador com conhecimento global do estado da rede simplifica o desenvolvimento de funcionalidades de gerenciamento. *(iv)* Redução de *vendor lock-in* possibilita maior integração de equipamentos de diferentes fabricantes.

Em SDN, os dispositivos de encaminhamento realizam operações elementares, baseadas em regras para determinar ações sobre os pacotes recebidos. Essas instruções são instaladas nos dispositivos pela interface *southbound*[1] (e.g., OpenFlow, ForCES). Nesse cenário, o plano de dados é representado pelos dispositivos de encaminhamentos que são interconectados entre si formando uma infraestrutura SDN. Já o plano de controle contém as camadas responsáveis pelas informações da infraestrutura de rede e possui uma série de elementos que programam o comportamento dos dispositivos de encaminhamento. Toda lógica de controle reside nesse plano. A interface *northbound* realiza a abstração das instruções de baixo nível utilizadas pela interface *southbound* e as oferecem como serviços ao plano de gerenciamento. O plano de gerenciamento contém o conjunto de aplicações que aproveitam as funcionalidades fornecidas

[1] *API* responsável pela comunicação com os dispositivos de encaminhamento de forma padronizada

pela interface *northbound* para fazer o controle lógico da rede (KREUTZ et al., 2015), tal como encaminhamento, *firewall*, balanceamento de carga e monitoramento.

Diferente das redes convencionais, SDN trabalha com políticas de encaminhamento de fluxos ao invés de pacotes. Um fluxo é uma sequência de pacotes que possuem características semelhantes, tais como endereços de origem, destino, identificadores de *Virtual Local Area Network* (VLAN) , protocolo, número da porta de origem e destino. As informações são combinadas compondo uma tupla que representa o fluxo em questão. Assim, os algoritmos de encaminhamento são aplicações executadas no plano de controle que utilizam essas tuplas com o objetivo de configurar os dispositivos SDN, para que estes realizem operações elementares de encaminhamento com base em regras de suas tabelas de fluxos, de modo a estabelecer a comunicação entre origem e destino.

1.1 OBJETIVO

Diante da motivação apresentada, o presente trabalho investiga o comportamento de algoritmos de encaminho de fluxos em SDN, realizando uma análise experimental comparativa. Para que o objetivo proposto neste trabalho seja realizado, os seguintes objetivos específicos devem ser atingidos:

- Investigar o paradigma SDN;
- Revisar algoritmos de encaminhamento existentes e seus objetivos;
- Estudar o funcionamento do emulador *Mininet* e do controlador *Floodlight*;
- Implementar os algoritmos de encaminhamento no controlador *Floodlight*;
- Definir as métricas para representar o desempenho da rede;
- Coletar dados experimentais; e
- Realizar uma análise dos resultados obtidos.

1.2 ESTRUTURA DO TRABALHO

O texto está estruturado da seguinte forma: O Capítulo 2 revisa os conceitos e tecnologias relacionadas com SDN. São revisadas as definições de fluxos de pacotes, OpenFlow, SDN, entre outros elementos necessários para contextualização do trabalho. O Capítulo 3 revisa de forma detalhada a estratégia de funcionamento das políticas de controle de fluxos escolhidas para o presente estudo. O Capítulo 4 faz uma revisão sobre o emulador utilizado para a simulação, apresentando junto a topologia da rede de múltiplos caminhos criada para o estudo das políticas. Este capítulo faz também uma breve introdução ao *software* para avaliação de desempenho de infraestruturas de comunicação. O capítulo 5 busca apresentar os resultados obtidos pelas

aplicações da ferramenta de avaliação utilizada para os testes. O capítulo 6 faz uma conclusão sobre o trabalho apontando possíveis desenvolvimentos futuros.

2 REVISÃO DE LITERATURA

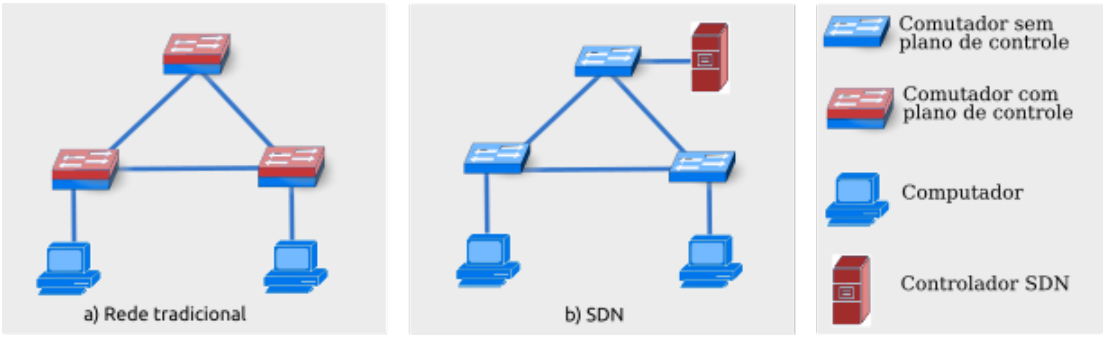
Este Capítulo apresenta uma revisão de literatura essencial para a compreensão do trabalho. Dentre os temas abordados, estão definições e arquitetura das SDNs (Seção 2.1). O funcionamento e principais componentes relacionados ao protocolo *OpenFlow* na configuração dos dispositivos (Seção 2.2). A arquitetura do controlador *floodlight* e o e funcionamento de aplicações suportadas (Seção 2.3). Por fim as considerações em relação aos conceitos apresentados para a compreensão do próximo capítulo é apresentado em (Seção 2.4)

2.1 REDES DEFINIDAS POR SOFTWARE

Em redes tradicionais, comutadores normalmente possuem sistemas fechados de gerenciamento e configuração disponibilizados pelos fornecedores dos dispositivos, dificultando a evolução de novos protocolos e tecnologias para gerenciamento de infraestruturas de comunicação. Essa característica, reduziu a flexibilidade da evolução arquitetural e dos protocolos (HANDLEY, 2006), principalmente devido a camada de controle e camada de dados embutidas em cada dispositivo de encaminhamento como mostrado na Figura 1 (a).

SDN é um paradigma emergente que possibilita mudanças para as limitações apresentadas em redes tradicionais. Em SDN, a separação do plano de controle e do plano de dados permite que os comutadores da rede se tornem simples dispositivos de encaminhamento, enquanto o controle gerencial da rede é centralizado em um controlador lógico chamado *Network Operation System* (NOS) (KREUTZ et al., 2015), facilitando a utilização de novas aplicações e protocolos (NUNES et al., 2014).

Figura 1 – Principal diferença entre *a)* redes tradicionais e *b)* SDN

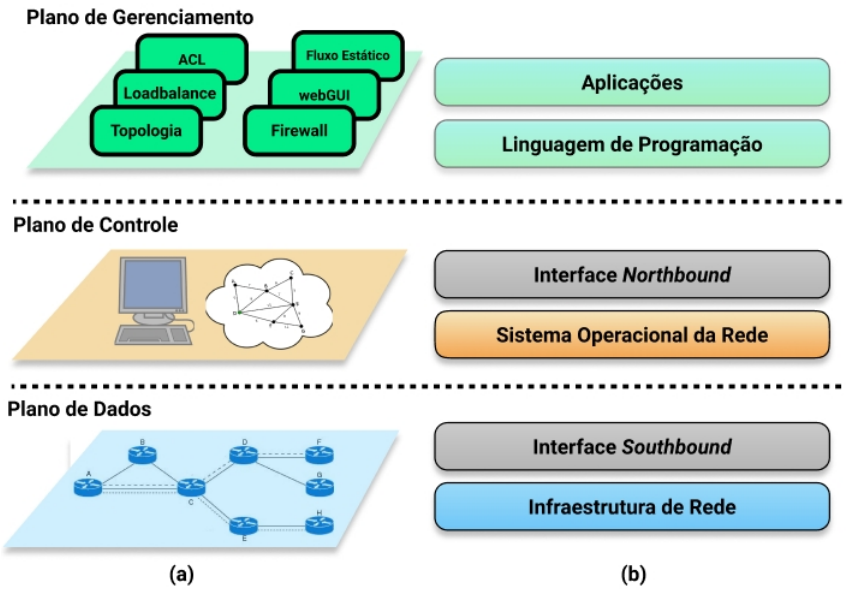


Fonte: Elaborada pelo autor (2021), com base em (NUNES et al., 2014)

A Figura 1 ilustra duas infraestruturas de rede. A primeira, *(a)* corresponde a uma rede tradicional, ou seja, a camada de controle está embutida nos comutadores, de modo a realizar o roteamento de pacotes utilizando uma visão local (parcial) da rede. A segunda *(b)*, ilustra uma infraestrutura SDN, seu plano de controle é separado em um ou mais controladores lógicos que utilizam a visão global da rede para efetuar o encaminhamento dos fluxos.

A abstração da camada de controle dos dispositivos em SDN permite que os comutadores atuem apenas como dispositivos de encaminhamento, baseado no conjunto de regras especificadas pelo controlador, conforme a política implementada. Um SDN pode ser apresentado como uma arquitetura de alto nível dividida em três principais planos funcionais, como ilustrado na Figura 2 (a), assim elencados: *(i)* Plano de Dados, envolve toda a infraestrutura da rede sendo composta por dispositivos de encaminhamento físicos ou virtuais, computadores e demais dispositivos terminais. *(ii)* Plano de controle, responsável por consolidar funcionalidades através de Interfaces de Programação de Aplicações (APIs) que supervisionam o comportamento de encaminhamento dos comutadores. *(iii)* Plano de gerenciamento, também conhecido como plano de aplicação, consiste em programas que fazem o uso das informações fornecidas pelo NOS para efetuar o gerenciamento da SDN (JARRAYA; MADI; DEBBABI, 2014).

Figura 2 – Representação de um SDN em a) planos, b) camadas.



Fonte: Elaborada pelo autor (2021), com base em (KREUTZ et al., 2015)

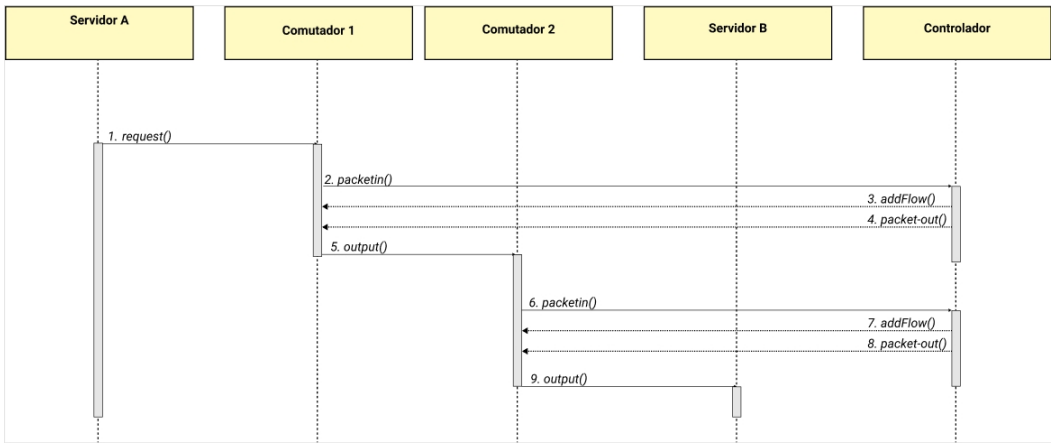
A Figura 2 é uma representação simplificada das camadas que constitui um SDN. Políticas de controle de fluxos podem ser implementadas como uma aplicação do plano de gerenciamento ou ainda como um serviço estendendo funcionalidades e módulos no próprio controlador. Na primeira existe a vantagem de se poder criar aplicações portáveis que podem ser utilizadas em diversos controladores. Já a segunda opção da mais agilidade por ser executada diretamente no controlador.

2.1.1 Fluxos de Dados

Um fluxo é uma sequência de pacotes que correspondem a uma determinada comunicação entre dispositivos finais. Por exemplo, pode-se considerar que *streaming* de vídeo realizado

de um computador A para um computador B é um fluxo de dados. Já um caminho de fluxo corresponde a um sub conjunto de dispositivos SDN que possuem regras em suas tabelas de modo que possam encaminhar os pacotes do fluxo de origem até o destino. No exemplo da Figura 3 um caminho de fluxo entre A e B é formado pelos comutadores 1 e 2 ligados de forma linear. O caminho de fluxo é calculado pelo controlador SDN e pode ser realizado de maneira proativa ou reativa, conforme a abordagem selecionada pelo controlador para satisfazer os requisitos do fluxo como *Quality of Service* (QoS), diminuição de latência (JR; FIORESE; KOSLOVSKI, 2016) ou melhor utilização da rede. A Figura 3, ilustra a ordem do processo de

Figura 3 – Diagrama de sequência, funcionamento da abordagem reativa.

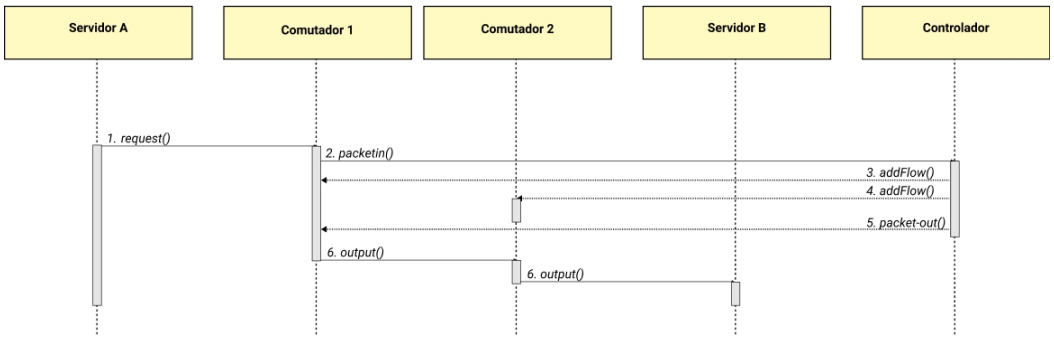


Fonte: Elaborada pelo autor (2021).

configuração dos dispositivos de rede utilizando a abordagem reativa. Neste cenário o servidor A envia um *Address Resolution Protocol* (ARP) para fechar uma conexão com o servidor B. Ao chegar no primeiro comutador, este não possui nenhuma regra para tratar o pacote gerando assim o primeiro *packet-in*, enviado ao controlador via canal seguro. O controlador verifica os campos do *packet-in* e analisa junto a topologia global da rede gerando regras de fluxos, essas regras contém as ações de encaminhamento adicionado pelo comando *addFlow*, criando em seguida um *packet-out* devolvendo o pacote ao comutador. Quando o pacote chega ao próximo comutador este gera outro *packet-in*, novamente o controlador adiciona as regras e em seguida devolve o pacote por um *packet-out*. O mesmo processo se repete a cada salto até que todos os comutadores do caminho sejam configurados.

Os demais pacotes que fazem parte do fluxo não geram *packet-in* devido às regras inseridas durante o evento *addFlow*. Este caminho permanece nos comutadores até que o *time-out* vinculado a regra ocorra, ou ainda, caso o controlador remova a regra por motivos específicos. Os pacotes inseridos pelo *addFlow* são similares aos do *packet-out*, porém o primeiro contém instruções em seu conjunto de ações que servem para inserir, remover e atualizar as regras, já o segundo apenas ações de encaminhamento.

Figura 4 – Diagrama de sequência, funcionamento da abordagem proativa.



Fonte: Elaborada pelo autor (2021).

Na abordagem proativa ilustrada pela Figura 4, ao receber o primeiro *packet-in* o controlador utilizara informações globais da rede para encontrar a lista de todos os dispositivos que compõem o caminho de A para B, conforme a política de encaminhamento implementada (*e.g.,* Menor quantidade de saltos, menor tráfego, entre outros). Ao obter essa lista o controlador inicia o processo de configuração dos dispositivos, enviando instruções com configuração específica para cada comutador que compõe o caminho do fluxo. Com os dispositivos configurados, os próximos pacotes do fluxo seguem segundo as regras existentes nas tabelas de fluxos, assim cada novo fluxo gera apenas um *packet-in*.

2.1.2 Plano de Gerenciamento

O plano de gerenciamento, ou plano de aplicações, interage com o controlador através da Interface de Programação de Aplicações (API) *northbound*, para manipular serviços oferecidos pela rede. Nesta camada, se destacam duas linhas de pesquisas relacionadas com o objetivo do presente trabalho: *(i)* aplicações SDN para gerenciar funcionalidades específicas da rede e *(ii)* aplicações SDN para gerenciar ambientes específicos. O primeiro interage com o controlador para gerenciar necessidades particulares da rede categorizadas em domínios de segurança, QoS, engenharia de tráfego e balanceamento de carga. No segundo caso, as aplicações SDN focam em ambientes específicos tais como computação em nuvem, rede móvel, virtualização de rede e *Network Functions Virtualization* (NFV) (JARRAYA; MADI; DEBBABI, 2014).

Uma SDN facilita o desenvolvimento de novas aplicações destinadas a segurança tal como *OpenFlow Random Host Mutation* (OFRHM), que possuem técnicas que escondem componentes da rede de *scanners* externos e internos que tentam descobrir alvos de possíveis ataques. Assim, endereços IPs virtuais são alocados dinamicamente de tal forma que os hospedeiros evitem a exposição de seus IPs reais que podem ser usados para ataques (JAFARIAN; AL-SHAER; DUAN, 2012). Aplicações específicas para migração de MVs (GHORBANI; CAESAR, 2012) e realização de IP *multicasting* (NAKAGAWA; HYOUDOU; SHIMIZU, 2012) são facilitadas com SDN. Neste segundo, o protocolo permite que o registro seja efetuado dinamicamente

permitindo que membros possam se juntar ou deixar o grupo de comunicação sem intervenção de gerenciamento da rede.

Ainda, aplicações de gerenciamento, como *Access Control List* (ACL), podem ser implementadas por um administrador. Essas aplicações permitem a definição de políticas para tomada de decisão em novos fluxos adicionados para ser verificada em conjuntos de regras que permitam determinadas conexões, por exemplo (um servidor somente pode utilizar o protocolo HTTP se estiver utilizando *proxy*). Complementarmente, aplicações para isolamento de redes também podem ser facilmente aplicadas, associando as portas que podem ser utilizadas por determinadas identificações de VLANs marcadas no cabeçalho do pacote, podendo também designar ações para que um pacote vindo de um determinado endereço físico ou IP seja marcado com uma identificação de VLAN. Em uma SDN aplicações de ACL contém regras que são consultadas durante um evento de *packet-in*, permitindo ou impedindo a criação do fluxo. Já as aplicações de *firewall* colocam essas regras nos comutadores permitindo o encaminhamento ou descartando o pacote através de uma ação *drop*. A maioria dos SDN utilizam endereçamento IP, no entanto, o *OpenFlow* permite que aplicações façam o gerenciamento da rede de modo a não utilizar de endereço IP (MCKEOWN et al., 2008), tais como utilizar apenas a camada de rede *Media Access Control* (MAC).

2.1.3 Plano de Controle

Denominado de NOS, é o responsável por fornecer as funcionalidades da rede. Ele contém o controlador que atua entre o plano de gerenciamento e o plano de dados (XIE et al., 2015). Sendo assim, o controlador tem acesso às funcionalidades fornecidas pelo *OpenFlow* da camada *southbound*, e as fornecem como serviços para as aplicações por meio da *interface northbound*.

2.1.3.1 Interface Northbound

É uma camada de API que abstrai instruções de baixo nível da camada *southbound* na configuração de entradas das tabelas de fluxos do plano de dados. A abstração pode ser feita através de linguagem de programação como *Frenetic* (FOSTER et al., 2011) e NetCore (MONSANTO et al., 2012). APIs abertas ou padronizadas são importantes, pois fornecem portabilidade às aplicações podendo ser utilizadas em diferentes NOS. Alguns controladores como Floodlight (FLOODLIGHT, 2012), NOX (GUDE et al., 2008) e OpenDaylight (MEDVED et al., 2014) possuem suas próprias *northbound* APIs específicas para seus propósitos. Nessa camada é comum a utilização da API RESTfull que faz o uso do *REpresentational State Transfer* (REST) utilizadas por diversos controladores. Efetuando assim o uso de protocolo HTTP/HTTPS para operações baseadas em *Uniform Resource Identifier* (URL). Dessa forma, pode-se utilizar dessas URLs para acessar informações dos dispositivos e manipular estatísticas e parâmetros específicos (ZHOU et al., 2014), similar aos comandos *Simple Network Management Protocol*

(SNMP) GET e PUT usados na gerência de redes tradicionais.

A escalabilidade dos controladores ocorre de acordo com a capacidade de processamento de fluxos do mesmo. Por exemplo, em computadores com 8 *cores*, um controlador pode manipular 1,6 milhões de fluxos por segundo o que os tornam capazes de trabalhar com altas taxas de fluxos e controlar redes de grande porte com facilidade (NUNES et al., 2014). Múltiplos controladores podem ser utilizados para reduzir a latência de comunicação na consulta entre os dispositivos de encaminhamento e os controladores e criar tolerância a faltas. Os modelos de controladores podem ser classificados em: centralizados e distribuídos.

2.1.3.2 Centralizados

Quando o SDN possui uma única entidade controladora trata-se de um controlador centralizado, naturalmente, apresenta um único ponto de falha interrompendo o funcionamento da rede na ocorrência de problemas. Controladores centralizados também podem ter limitação quanto a escalabilidade de elementos em redes com altas densidades de dispositivos de encaminhamento (KREUTZ et al., 2015). Os controladores centralizados mais populares são:

- **Beacon:** Controlador desenvolvido pela universidade de *Stanford* baseado na linguagem de programação Java. Possui uma estrutura modular que permite atualizações em tempo de execução sem causar interrupção na rede, além de explorar o protocolo *OpenFlow* com foco na facilidade de desenvolvimento, alto desempenho, iniciar e parar aplicações em tempo real. Executando em uma máquina de 12 *cores*, pode suportar até 12.8 milhões de fluxos (ERICKSON, 2013).

- **Floodlight:** É um controlador *open source* amplamente utilizado, baseado totalmente em Java e distribuído pela licença Apache. Sua arquitetura é baseada em *multi-threads* permitindo paralelismo em processadores de múltiplos núcleos. É um projeto que foi originado no controlador *Beacon* e todos os elementos são módulos que exportam serviços. Permite descobrir a topologia automaticamente e pode ser usado em redes com comutadores *OpenFlow* e não *OpenFlow*, simultaneamente, além de funcionar com a ferramenta para emulação de rede. Atualmente, é mantido com suporte de empresas como *Intel*, *Cisco*, *HP*, e IBM (FLOODLIGHT, 2012). O *Floodlight* é tratado com mais detalhes na Seção 2.3.

- **NOX:** Controlador *OpenFlow* que não suporta paralelismo, utilizado principalmente para pesquisas por ser um NOS simples, que fornece primitivas para o gerenciamento de eventos e funções de comunicação de comutadores (GUDE et al., 2008).

- **POX:** Este é um controlador centralizado desenvolvido para ser o sucessor do *NOX*. Ele foi desenvolvido em *python* e seu projeto foi desenvolvido de modo que ele seja fácil para instalar e executar.

- **RYU:** É um controlador *open source* desenvolvido e mantido pela *Nippon Telegraph and Telephone* (NTT) na linguagem de programação *python* e suporta os protocolos *OpenFlow*, *Netconf* e *OF-confg*. A arquitetura do controlador é dividida em três principais camadas: 1) camada de aplicação envolve lógicas de gerência do plano de dados; 2) serviços da SDN e 3) configurações e instruções do plano de dados (TELEGRAPH, 2018).

2.1.3.3 Distribuídos

SDN com NOS distribuídos utilizam as APIs *east/westbound* para que os controladores possam se comunicar e sincronizar dados da rede entre si. Por possuir normalmente mais de um único controlador, apresentam maior tolerância a falhas. Exemplos de controladores distribuídos são:

- **DISCO:** é um controlador utilizado para redes de multi-domínio, interconectando *data-centers* dispersos geograficamente, como redes corporativas e acadêmicas, deixando a rede resiliente, escalável e extensível. É um controlador aberto, capaz de suportar planos de dados heterogêneos e distribuídos sobrepostas em redes de longa distância (PHEMIUS; BOUET; LEGUAY, 2014).

- **ONOS:** *Open Network Operating System* é um controlador distribuído experimental e sua arquitetura é formada por uma instância primária e diversas secundárias, aptas a virar instâncias primárias caso esta entre em falha. Nessa arquitetura, um comutador pode se conectar a diversas instâncias do ONOS porém apenas uma será primária (BERDE et al., 2014).

- **OpenDaylight:** projeto *open source* hospedado pela *Linux Foundation*. Atualmente esta na versão *aluminium* lançada em 2020. Além do padrão OpenFlow ele suporta tecnologias emergentes para realização de gerenciamento e programabilidade da rede baseadas em modelo como NETCONF/YANG (MEDVED et al., 2014). Por se tratar de um projeto que envolve diversas empresas (*e.g, Cisco , Citrix , IBM , Juniper Networks e outras*) este controlador consolidou a adesão de vários projetos, como OpenFlow, BGP, NETCONF, OVSDB com tecnologias dos diversos contribuidores.

2.1.4 Plano de Dados

É o responsável pelo encaminhamento dos fluxos, composto por comutadores e roteadores que atuam como equipamentos de encaminhamento. Cada dispositivo possui uma série de tabelas de fluxo nos quais o controlador insere e remove regras e ações para tratar os pacotes. Para que esses comutadores possam analisar grandes quantidades de regras associadas a muitos fluxos os dispositivos contam com uma memória *Ternary Content-Addressable Memory* (TCAM) que permite três tipos de comparações resultantes em (*e.g.,* 0, 1 e *) no qual o 0 significa que o campo comparado deu falso, 1 significa que o campo correspondeu a regra e "* "é um curinga

que resulta sempre em expressão verdadeira para qualquer valor contido no campo do cabeçalho. O plano de dados é composto por duas camadas: interface *southbound* e infraestrutura de rede.

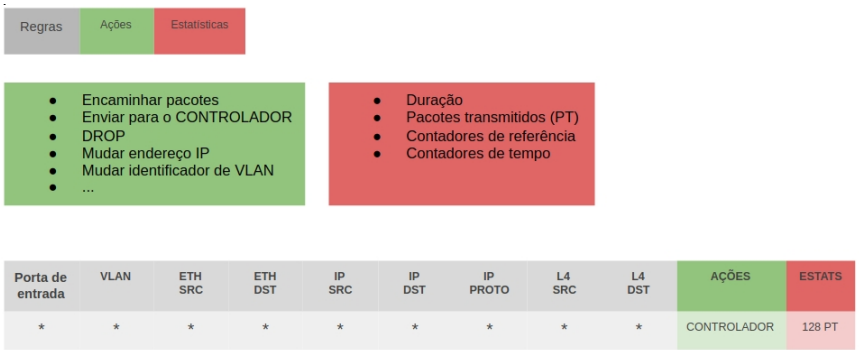
2.1.4.1 Interface Southbound

A *Southbound* é uma ponte de conexão entre o controlador e os dispositivos de encaminhamento, separando as funcionalidades do plano de controle e o plano de dados. Esta interface pode ser vista como serviços de *drivers* que fornecem um conjunto comum de funcionalidades dos dispositivos de encaminhamento para as camadas de controle do SDN. Existem diversas APIs com este propósito, como ForCES (DORIA et al., 2010), OVSDB (PFAFF; DAVIE, 2013) para configuração de comutadores virtuais, POF (SONG, 2013) e OpFlex (SMITH et al., 2014). Uma das interfaces mais populares aceitas pela academia e indústria é o *OpenFlow* explicado em detalhes na Seção 2.2.

2.1.4.2 Infraestrutura

A infraestrutura de um SDN é composta por dispositivos que tem como principal funcionalidade encaminhar fluxos, apenas realizando operações correspondentes as entradas das tabelas de fluxos. Atualmente é possível utilizar dois tipos de comutadores: Os baseados em *software*, no qual se destaca o *Open vSwitch*, que possui uma implementação com um elevado grau de flexibilidade (PETTIT et al., 2010) permitindo a criação de múltiplas instâncias com diversas interfaces de rede virtuais em um único computador; e os comutadores *OpenFlow* baseados em *hardware*, projetados com *Application-Specific Integrated Circuits* (ASIC) especificamente para atender os requisitos das SDN, como alta taxa de transferência e capacidade elevada de portas, porém não apresentam tanta flexibilidade de novas implementações quanto os baseados em *software*.

Figura 5 – Exemplo de uma entrada da tabela de fluxo.



Fonte: Elaborada pelo autor (2021), com base em (DORGIVAL et al., 2012)

As Figuras 5 ilustra as informações da tabela para o encaminhamento de fluxos em SDN. Como a decisão de encaminhamento é feita pelo controlador, então cabe a ele atualizar estas

entradas podendo utilizar para encaminhamento tanto partes do cabeçalho que correspondem a camada de rede *L2* ou a camada transporte *L3*, além de poder utilizar a camada *L4* para criação de regras de *firewall* e controle de acesso. Quando um pacote chega no comutador é realizado uma comparação entre cada campo do cabeçalho com seu respectivo campo da tabela entrada. Além das regras utilizadas para verificação, a tabela possui outros dois elementos: ações e estatísticas. De acordo com comparações realizada pelas regras de fluxos, os pacotes são processados conforme as ações de cada regra. As estatísticas são utilizadas para contabilizar os fluxo, tais como quantidade de pacotes, *timeout*[1], *cookie*[2] e prioridade [3].

Comutadores comerciais para SDN utilizam circuitos ASIC para realizar processamentos de tarefas específicas como análise de pacotes, *buffering*, escalonamento e encaminhamento de pacotes. No entanto, apesar de avançados, ainda existem desafios para o armazenamento da grande quantidade de fluxos em tabelas e processamento em picos inesperados de tráfegos. Para resolução de tais problemas, arquiteturas de equipamentos ASIC usam Unidade Central de Processamento (CPU). Dessa forma, quando um novo fluxo é adicionado, a CPU verifica a existência de fluxos inativos que estejam sem receber pacotes em um determinado período de tempo pelo campo *timeout*, substituindo essa entrada da tabela pelo novo fluxo, passando ele para ser processado pela ASIC. Caso contrário, a própria CPU fica responsável por encaminhar os novos fluxos dessa categoria. Da mesma forma, a Memória de Acesso Randômico Dinâmica (DRAM) pode ser utilizada para armazenamento de *buffer* em caso de picos de tráfegos, para melhorar a eficiência no encaminhamento de pacotes (NUNES et al., 2014) (GONG et al., 2015). A CPU pode direcionar fluxos maiores para serem processados pela ASIC enquanto os fluxos menores ficam sendo totalmente processados pela CPU.

2.2 PADRÃO OPENFLOW

Dentre os padrões ofertados no mercado para estabelecer uma comunicação entre os comutadores e os controladores, o que mais vem se destacado ao longo dos últimos anos é o *OpenFlow*. Atualmente se encontra na versão 1.5 sendo a versão 1.3 mais amplamente usada em comutadores comerciais, esse padrão vem sendo desenvolvido e mantido pela *Open Network Foundation* (ONF) permitindo que cada vez mais novas possibilidades de aplicações sejam desenvolvidas. O padrão OpenFlow é caracterizado por três componestes: tabelas de fluxos, canal seguro e protocolo OpenFlow.

---

[1] *Timeout* é um marcador utilizado para determinar quanto tempo essa regra pode durar sem ser utilizada, até se tornar inválida.

[2] Cookie é um identificador 64 bits que serve para mapear e armazenar informações relevantes ao controlador sobre determinada regra, muitas regras podem compartilhar do mesmo cookie.

[3] Prioridade serve para determinar a ordem de precedência que as regras devem ser aplicadas.

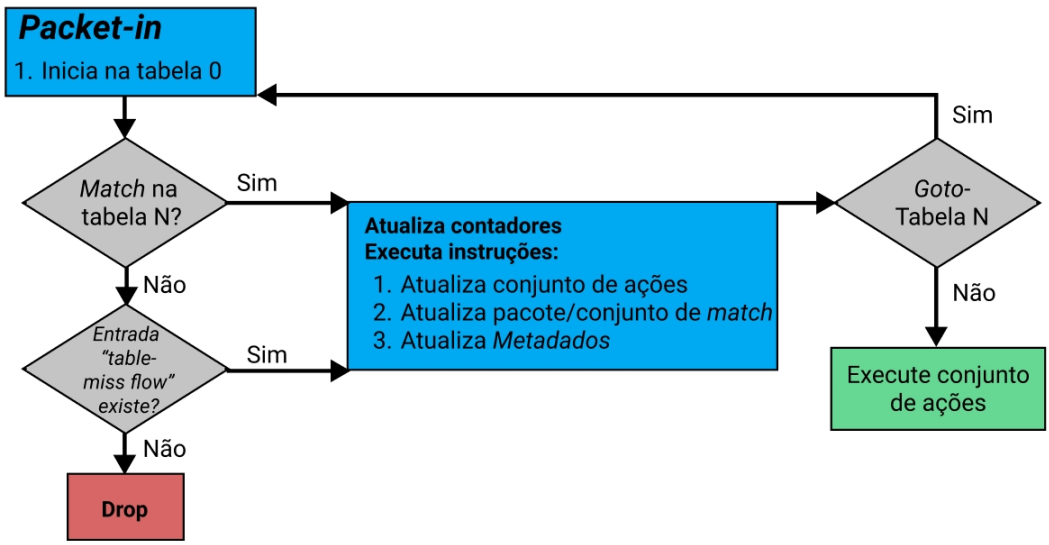
2.2.1 Tabelas de Fluxos

As tabelas de fluxos são encontradas nos dispositivos de encaminhamento, no qual cada entrada da tabela conta com três principais partes: i) cabeçalho, ii) contadores e iii) ações. Os campos de cabeçalho são utilizados para realizar a verificação do fluxo e assim determinar se estes pertences a determinada regra (*match*). O *OpenFlow* 1.3 conta com 44 campos de cabeçalhos que podem ser utilizados para a formação de um *match* (NYGREN et al., 2015). Os campos de contadores armazenam informações para manter o controlador atualizado sobre o status da rede. As ações determinam como o pacote do fluxo será tratado quando um *match* ocorrer.

2.2.1.1 Cabeçalho

Quando é inserido em um comutador uma entrada na tabela de fluxo que irá gerar um *match* o controlador pode informar um curinga (*ANY*) para os campos que devem corresponder a qualquer valor contido no respectivo campo do pacote. Exemplo, quando o curinga é informado no campo de *MAC_DST*, significa que qualquer endereço MAC irá corresponder nesse campo. Para ocorrer um *match* o cabeçalho do pacote deve corresponder ao cabeçalho da entrada da tabela de fluxo, exceto os curingas. Caso não haja entradas que formam um *match* com regras específicas, o *OpenFlow* permite que este pacote seja tratado de três formas. Na primeira com a criação de uma regra chamada *table-miss flow*, entrada específica na tabela com prioridade 0 e todos os campos curingas com uma ação de enviar ao controlador. Caso o comutador esteja ligado ao controlador através de outro comutador, é preciso configurar essa regra de modo que a ação da *table-miss flow* envie para a porta vinculada ao segundo comutador intermediário.

Figura 6 – Exemplo de processamento interno de fluxo em um comutador OpenFlow 1.3



Fonte: Elaborada pelo autor (2021), com base de (ONF, 2012)

A segunda abordagem é apenas descartar o pacote com a ação *drop*, assim o SDN trabalhará só com fluxos premeditados, e todos os pacotes desconhecidos são descartados. A terceira forma é a utilização de *goto*[4] pulando para análise de regras nas próximas tabelas.

Como ilustra a Figura 6, todos os pacotes que chegam ao comutador passam por uma série de processamento iniciando na tabela 0 com campos do *pipeline* inicializados. Quando ocorre um *match* entre uma das entradas da tabela e o cabeçalho desse pacote é avançado a *pipeline* para a fase de atualização dos contadores dessa regra, aplica o conjunto de instruções e o conjunto de ações. Esse processo pode ocorrer diversas vezes até que as ações de todas as entradas que causaram *match* sejam aplicadas. Caso contenha ações de *output* o comutador pode gerar o pacote de saída definido pelas ações, ou ainda consultar as próximas tabelas de saídas em um novo *pipeline* de processamento. Pacotes enviados pelo controlador podem utilizar somente da *pipeline* que executa instruções, adicionando e removendo entradas ou coletando dados armazenados pelos contadores.

2.2.1.2 Contadores

Os contadores são utilizados para armazenar informações relevantes aos fluxos. A tabela 1 contem os contadores do OpenFlow. Esses são atualizados durante a *pipeline* de processamento dos pacotes e podem fornecer estatísticas de modo que o controlador possa efetuar o melhor gerenciamento da rede, como exemplo, utilizar os contadores de fluxo, *bytes* recebidos e duração para calcular a taxa de transferência relacionada ao fluxo. Assim o controlador pode identificar e reescalonar os fluxos para filas de menor ou maior vazão dependendo de políticas implantadas, ou ainda calcular custo de enlaces dinâmicos pelas estatísticas de portas. Os contadores de duração são obrigatórios e utilizam a precisão dos segundos, estes são inicializados no momento em que o fluxo é inserido na tabela, e comparados aos campos de *time-out* para determinar que uma regra tenha perdido a validade. Quando a tabela de fluxo esta totalmente utilizada não possuindo mais entradas vazias, os contadores são utilizados para que o próprio comutador decida qual entrada deve ser substituída, evitando assim que esses erros consumam tempo do controlador que possui uma lista de tarefas mais prioritárias em andamento.

2.2.1.3 Instruções

O conjunto instruções estão vinculados as entradas da tabela de fluxo e servem para realizar alterações em campos do cabeçalho do pacote, alterações no conjunto de ações e alterações da *pipeline* de processamento. O OpenFlow possui seis tipos de instruções que estão listadas na Tabela 2 em ordem de precedência. É através das instruções que o controlador insere e remove as regras de *match*. Instruções do tipo *goto* são sempre executadas por último.

[4] goto é uma instrução utilizada para definir a próxima tabela que deve ser consultada, essa instrução pode apenas avançar o número da tabela.

Tabela 1 – Lista de contadores OpenFlow.

Contador	Bits	Contador	Bits
Por Tabela de Fluxo		Por grupo	
Entradas ativas	32	entradas ativas	
Consulta de pacotes	64	Contador de pacotes	
Pacotes correspondidos	64	Contador de bytes	
Por entrada de fluxo		Duração	
Pacotes recebidos	64	Por métrica	
Bytes recebidos	64	Contagem de fluxo	32
Duração	32	Contagem de pacotes input	64
Por porta		Contagem de bites de input	64
Pacotes recebidos	64	Duração (segundos)	
Pacotes transmitidos	64	Por métrica de banda	
Colisão	64	Contagem de pacotes	64
Erros transmitidos	64	Contagem de bytes	64
Pacotes recebidos e descartados	64	Por Fila	
Pacotes transmitidos e descartados	64	Pacotes transmitidos	64
Erros recebidos	64	Bytes transmitidos	64
Duração (segundos)	32	Duração (segundos)	32

Fonte: Elaborada pelo autor (2021), com base em (ONF, 2012)

Tabela 2 – Lista de instruções do OpenFlow.

Instrução	Parâmetro	Descrição
Aplica-Ações	Lista de ações	Aplica o conjunto de ações vinculada a entrada imediatamente.
Limpa Ações		Limpa o conjunto de ações imediatamente.
Escreva-Ações	Lista de ações	Realiza um merge entre a lista de ações atual e a lista de ações passada por parâmetro. Quando uma ação de mesmo tipo ja exista então sobrescreve a ação. Caso não exista, adiciona.
Escreva-Metadados	Metadados/ Mask	Insere valor no metadado de acordo com os bits da mascára.
Medidor	Limiar de estatísticas	Encaminha o pacote a algum medidor, podendo descartar pacotes caso atinja o limiar.
Goto-Tabela	Id da tabela	Indica o id da próxima tabela que deve ser processada durante o pipeline de processamento.

Fonte: Elaborada pelo autor (2021), com base em (ONF, 2012)

2.2.1.4 Ações

As ações determinam como o dispositivo tratará o fluxo. O conjunto de ações relacionadas a cada entrada da tabela de fluxo é por padrão vazio e são inseridas durante a *pipeline* de

processamento das instruções (*e.g.,* Aplica-Ações, Escreva-Ações, *Goto*). Durante o processamento, apenas uma ação de cada tipo pode ser adicionada a lista. Exemplo, quando adicionado ao conjunto de ações uma ação de atualizar a porta *Transmission Control Protocol* (TCP) para outro valor, e outra ação realiza a mesma operação para um valor diferente, esta ação não pode ser inserida a lista de ações, pois apenas uma é permitida. Conforme o processamento vai avançando entre as tabelas com o uso do *goto*, novas ações são inseridas ao conjunto (NYGREN et al., 2015). O padrão OpenFlow permite onze tipos de ações, das quais apenas três primeiras são obrigatórias, seis das comumente utilizadas são:

1 *Output* é a ação que encaminha o pacote para uma porta específica, como ilustrado na figura 6 quando esta ação é realizada na *pipeline* com um conjunto de ações, os pacotes são clonados passam pelo mesmo procedimento. Assim cada cópia é enviada por uma porta.

2 Grupo, processa o pacote de acordo com ações determinadas aos seus respectivos identificadores de grupos.

3 *Drop*, ação responsável por descartar o pacote.

4 Enfileirar, ação que insere o pacote na fila de uma porta configurada conforme políticas de QoS. Uma única porta pode conter diversas filas e cada fila pode ser configurada com sua própria largura de banda. As filas descartam pacotes para manter a largura de banda em seu limite máximo, por exemplo, uma fila configurada para 1 Mb/s descartará pacotes de modo a manter o fluxo em 1 Mb/s. Cada porta pode conter múltiplas filas em diferentes configurações, permitindo o controlador restringir conforme a necessidade.

5 Medidor, direciona o pacote para ser tratado em uma tabela de medidores. Nessa tabela os medidores podem definir limitante de taxas, QoS, largura de banda, entre outros medidores inseridos pelo controlador. Dependendo do resultado desse processamento o pacote pode ser descartado, para se ajustar aos parâmetros do medidor.

6 Copiar campo, ação que pode atualizar os valores de algum campo do pacote. Aplicações de balanceamento de carga de servidores, podem utilizar essa ação para que múltiplos servidores possuam o mesmo endereço IP, distribuindo assim as requisições.

2.2.2 Canal Seguro

Também conhecido por canal OpenFlow, este tem o objetivo de conectar cada comutador ao controlador. Por meio deste canal o controlador gerencia as regras das tabelas de fluxos, coleta estatísticas e recebe pacotes *packet-in*. Como uma SDN pode possuir múltiplos dispositivos de encaminhamento, nem sempre é possível criar enlaces diretos entre dispositivos de encaminhamento e controlador, sendo assim, a topologia de rede compartilhada por diferentes usuários também é usada para efetuar o gerenciamento da infraestrutura. Dessa forma toda comunicação

realizada entre dispositivos e controlador normalmente são encriptadas utilizando *Secure Socket Layer* (SSL) e *Transport Layer Security* (TLS) para garantir a confidencialidade da comunicação (PFAFF et al., 2012).

2.2.3 Mensagens e Exemplos

O protocolo OpenFlow é uma das principais partes do padrão. Neste protocolo são utilizados três categorias de mensagens: *(i) Controlador-Comutador*, são mensagens iniciadas pelo controlador com o objetivo de analisar e gerenciar os estados do comutador. *(ii)* Assíncronas, são mensagens geradas pelo comutador para atualizar o controlador sobre informações de eventos de rede que causam mudanças de estados do dispositivo. *(iii)* Simétricas, mensagens que podem ser geradas tanto por dispositivo quanto pelo controlador sendo enviadas sem solicitação, utilizadas para inicialização do dispositivo e verificação do canal seguro. (PFAFF et al., 2012).

Figura 7 – Tabela de fluxo com regras de encaminhamento, *firewall* e comunicação por VLANs.

Porta	VLAN	ETH SRC	ETH DST	IP SRC	IP DST	IP PROT	L4 SRC	L4 DST	Ações
*	*	*	10-78-D2-DA-24-83	*	*	*	*	*	Porta 4
*	*	*	*	*	*	TCP	*	22	DROP
*	1	*	10-78-D2-DA-24-83	*	*	*	*	*	Porta 4,6,9

Fonte: Elaborada pelo autor (2021), com base em (DORGIVAL et al., 2012)

A Figura 7 contém as principais tuplas que realizam operações sobre o cabeçalho de cada pacote do fluxo, executando assim as ações correspondentes e armazenando estatísticas. O * indica o curinga *don't care*, ou seja, quando o cabeçalho do pacote *OpenFlow* utilizar a entrada da tabela para verificar se são iguais, os campos representados na tabela por * resultam em expressão verdadeira para qualquer valor. Segindo essa lógica, a primeira linha da tabela especifica que todos os pacotes que entrarem neste comutador com o endereço de MAC destino igual a *10:78:D2:DA:24:83*, serão encaminhados pela porta física 4. Qualquer pacote que utilizar o protocolo TCP/IP com destino a porta 22 será descartado, ou seja, ele atua como um *firewall* bloqueando qualquer comunicação *Secure Shell Protocol* (SSH). Já a última regra da tabela de fluxo informa que qualquer pacote com a *tag* da VLAN 1 e destino ao endereço físico *10:78:D2:DA:24:83* terá cópias mandadas para as portas 4, 6 e 9 simultaneamente (DORGIVAL et al., 2012).

As principais mensagens suportadas pelo protocolo são apresentadas na Tabela 3. As mensagens como *Flow-Removed* funcionam da seguinte forma: o controlador adiciona um marcador no momento de criação do *match*, assim indica ao comutador que notifique o controlador quando a regra expirar, seja por tempo ocioso do fluxo, ou por por tempo limite. Essa mensagem é útil para manter o controlador atualizado sobre as configurações dos fluxos sem que haja a

Tabela 3 – Principais classes de mensagens suportadas pelo protocolo OpenFlow.

MENSAGENS DO PROTOCOLO OPENFLOW	
Controlador para Comutador	
Características	Solicita as capacidades do dispositivo de encaminhamento.
Configuração	Estabelecer e solicitar parâmetros de configuração do dispositivo.
Modify-State	Mensagens geradas pelo controlador para efetuar gerenciamento do estado dos comutadores. Dessa forma é possível atualizar inserir e remover tabelas de fluxos no dispositivo
Read-State	Mensagem de solicitação de informações dos comutadores, estatísticas, configurações e capacidade.
Packet-out	Mensagem *packet-in* devolvida, contendo ações para aplicação imediata do comutador, contém o número da porta de saída. Essa mensagem também pode contém as ações de filas e medidores.
Role-Request	Mensagem utilizada para configurar ou obter configuração do canal *OpenFlow*, usado em caso de vários controladores, para especificar qual deve ser usado como primário.
Assíncronas	
Packet-in	Mensagem enviada quando um fluxo não identificado entra no comutador, pode enviar o pacote completo ou apanas parte de seu cabeçalho, mantendo o pacote armazenado na memória para prosseguir com o encaminhamento de acordo com a resposta do controlador.
Flow-Removed	Informam o controlador quando um fluxo é removido da tabela quando seu limite de tempo é excedido ou caso a quantidade pacotes pertencente ao fluxo seja atingida.
Port-status	Informa o controlador de mudanças no estado da porta caso ela seja desativada por mensagens de configuração ou por apresentar falhas no enlace.
Error	Informa o controlador quando o comutador apresenta algum erro.
Simétricas	
Hello	Mensagens enviadas por controladores e comutadores quando a rede é iniciada.
Echo	Mensagem do tipo requisição e resposta que podem ser enviadas pelos comutadores ou pelo controlador para verificar problemas em enlaces ou no canal seguro.
Experimenter	Mensagem que fornece um padrão para os comutadores *OpenFlow* oferecerem funcionalidades adicionais.

Fonte: Elaborada pelo autor (2021), com base em (ONF, 2012)

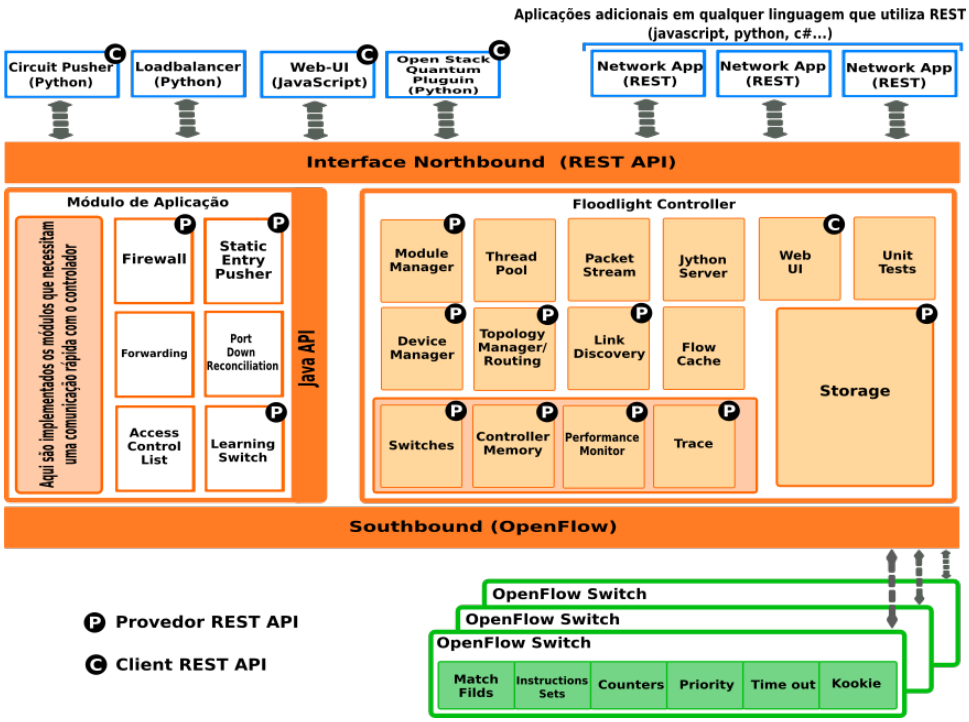
necessidade de solicitações de consultas aos comutadores. Já que ao inserir os fluxos ele pode memorizar em caches, e removê-los com esses eventos.

2.3 CONTROLADOR FLOODLIGHT

O *Floodlight* é um controlador centralizado que utiliza o protocolo *OpenFlow*, possuindo suporte a *multi-threading*, implementado na linguagem *Java*. Ainda oferece através de módulos e

aplicações que atuam no gerenciamento da rede. Dentre as opções de controladores investigadas, *Floodlight* foi identificado como o controlador centralizado com código estável e comunidade de desenvolvimento ativa. Esses requisitos são essenciais para o desenvolvimento do presente trabalho. Ainda, *Floodlight* possui total integração com o ambiente de prototipação *Mininet* (detalhado na Seção 4.1).

Figura 8 – Arquitetura modular interna do controlador *Floodlight*.



Fonte: Elaborada pelo autor (2021), com base em (FLOODLIGHT, 2012)

Sempre que o *Floodlight* é executado, todos os módulos de aplicações também são inicializados, assim as APIs REST expostas pelos módulos ficam acessíveis pela porta 8080, dessa forma todas as aplicações REST seja qual for a linguagem pode recuperar informações e invocar serviços através de URLs.

2.3.1 Aplicações Disponibilizadas

A Figura 8 ilustra a arquitetura atual do Floodlight. Em azul ficam as aplicações independentes de linguagem de programação que fazem requisições REST. Em módulo de Aplicação ficam as aplicações que necessitam comunicações rápidas com o controlador, objetos compartilhados como é o caso da topologia, lista de enlaces, lista de comutadores. Toda informação do controlador é acessível importando as classes de *Floodlight Controller* e implementando as *interfaces*. Os módulos marcados com "*p*", indicam que estes além de poderem ser instanciados por outros módulos podem ser acessados via URLs. Dentre os módulos e aplicações apresentados

pela figura as principais (relacionadas com o presente trabalho) são apresentadas com maiores detalhes nos tópicos:

- *Firewall*: Aplicação para gerenciar regras de permitir/negar tráfegos específicos ou combinações de regras. Possui um conjunto de APIs REST para adicionar e remover regras de Firewall aplicadas nos comutadores.

- *Port Down Reconciliation*: Módulo específico para procurar e excluir regras das tabelas de fluxos relacionadas com enlaces que estejam apresentando falhas. Dessa forma o controlador permite reavaliar o caminho e atualizar a topologia de rede, evitando esses enlaces. Este módulo pode gerar problemas em topologias emuladas de grande porte, pois a demora de escalonamento do processador pode gerar *time-out* fazendo com que as regras sejam removidas inesperadamente.

- *Forwarding*: Este módulo é responsável por tratar as mensagens enviadas pelos comutadores (*e.g., Packet-in*, *Flow-removed*, *Port-status*) e processar as informações das mensagens gerando respostas aos comutadores, como *Add-flow* e *Packet-out*. Este módulo recebe uma *thread* para cada mensagem recebida, entregue conforme o escalonamento de serviços internos do controlador. Por padrão este módulo conduz o encaminhamento proativo e foi utilizado como base na construção das políticas de *Round-Robin*, caminho mais curto reativo e caminho de menor tráfego explicados em detalhes no Capítulo 3.

- *Static Flow Entry Pusher*: é uma aplicação para instalar entradas de fluxos nos comutadores manualmente através de URLs. Com um conjunto de APIs REST para inserir remover e atualizar as entradas de cada comutador, permitindo a integração com outras linguagens de programação como o *Python*. A aplicação *Circuit Pusher* ilustrado na Figura 8 faz o uso desse módulo para gerar caminhos estáticos entre origem e destino.

2.3.2 Arquitetura do Controlador

Ilustrado na Figura 8, o núcleo do *Floodlight* apresenta uma arquitetura modular, para implementar tanto as funcionalidades quanto as aplicações que utilizam a API REST. Entre os existentes módulos do núcleo do controlador estão alguns responsáveis pelo funcionamento e atualização das informações de desempenho e utilização da rede. Os componentes principais da arquitetura são:

- *Module Manager*: módulo responsável pela inicialização dos serviços que estão descritos no arquivo de propriedades.

- *Thread Pool*: é um *wrapper* para *ScheduledExecutorService* que faz o controle de tempo de execução das *threads* de modo periódico ou específico dependendo do serviço.

- *Packet Streamer*: é um serviço de *streaming* de pacotes que são trocados entre dispositivos de encaminhamento e o controlador.
- *Floodlight Provider*: possui duas funcionalidades principais, sendo a primeira manusear a conexão entre os comutadores transformando mensagens OpenFlow em eventos que possam ser lidos pelos demais módulos do controlador. A segunda funcionalidade é despachar a mensagem para o módulo responsável pelo tratamento do evento.
- *Device Manager*: faz o registro de adição e remoção mantendo atualizados os dados de dispositivos terminais da rede, tal como computadores.
- *Link Discovery*: módulo responsável por manter o *status* dos enlaces da rede *OpenFlow* atualizados, este módulo dispara pacotes para serem enviados por portas do comutador e assim verificar enlaces ativos e inativos.
- *Topology Service*: responsável por atualizar as informações da topologia da rede. A topologia pode ser instanciada em qualquer novo módulo para obtenção de informações de comutadores e enlaces, permitindo assim a visão global da infraestrutura.
- *REST API Server*: este módulo é responsável por expor a API REST através de HTTP.
- *Flow Cache*: módulo responsável por tratar eventos de rede, tal como manusear fluxos em caso de falha no enlace.

Alguns módulos da arquitetura não são abordados em detalhes, pois não contribuem diretamente para o desenvolvimento de políticas de encaminhamento.

2.4 CONSIDERAÇÕES PARCIAIS

Este capítulo faz uma revisão de literatura sobre SDN, detalhando seus principais componentes arquiteturais, focando na definição de fluxos e atuação dos controladores. Em um segundo momento, uma revisão foi apresentada sobre o protocolo *OpenFlow*, um padrão aberto para configuração dos dispositivos em SDN. *OpenFlow* oferece ao controlador três principais funcionalidades, sendo elas: mensagens de eventos, estatísticas da rede, e requisições de *packet-in* utilizadas quando houver falta na entrada das tabelas de fluxos dos dispositivos de encaminhamento.

Dentre os diversos controladores apresentados está o Floodlight, escrito na linguagem de programação *Java* utiliza o protocolo *OpenFlow* para configuração dos dispositivos de encaminhamento. Por ser o controlador alvo do presente trabalho, a arquitetura do Floodlight foi discutida em detalhes.

O próximo Capítulo 3 apresenta os quatro principais algoritmos aplicados para realizar o encaminhamento dos fluxos em SDN. Dois deles utilizam o caminho mais curto, um de abordagem reativa e outro proativa, em que visam o caminho de menor quantidade de saltos. Os

outros dois algoritmos fazem o uso de múltiplos caminhos de acordo com estratégias próprias para seleção de rotas alternativas e balanceamento do tráfego.

3 ALGORITMOS PARA ENCAMINHAMENTO DE FLUXOS

Dentre as categorias de atuação das aplicações de gerenciamento SDN, o presente trabalho está contido na categoria de Engenharia de Tráfego. Com o paradigma SDN, novos requisitos para engenharia de tráfego foram introduzidos utilizando a visão global da rede para desenvolvimento de algoritmos de encaminhamento de fluxo. Esta categoria é importante para otimizar o desempenho da infraestrutura de rede, utilizando análises dinâmicas, predição e regulagem do comportamento da rede (AKYILDIZ et al., 2014). Assim é possível maximizar a utilização da rede, minimizar o consumo de energia, efetuar o balanceamento de carga entre outros, permitindo que o administrador tenha maior controle sobre o modo como a rede opera (BEZERRA et al., 2016).

Em infraestruturas com topologias de múltiplos caminhos, as aplicações de gerenciamento de tráfego pode utilizar os dados sobre a rede e efetuar a execução de diferentes estratégias para políticas de encaminhamento de fluxo. Com isso o presente Capítulo é dividido em: na Seção 3.1 é feito uma revisão sobre encaminhamento de fluxos. A Seção 3.2 contém o funcionamento em detalhes da política de encaminhamento *Round-Robin* (RR), que utiliza múltiplos caminhos para o gerenciamento de rotas. A Seção 3.3 explica o funcionamento da política Caminho Mais Curto Reativo (CMCR). A Seção 3.4 contém os detalhes de funcionamento da política Caminho Mais Curto Proativo (CMCP). O funcionamento da política de Caminho de Menor Tráfego (CMT) está descrito na Seção 3.5. A Seção 3.6 descreve os trabalhos relacionados e a Seção 3.7 as considerações parciais do Capítulo 3.

3.1 ENCAMINHAMENTO DE FLUXOS

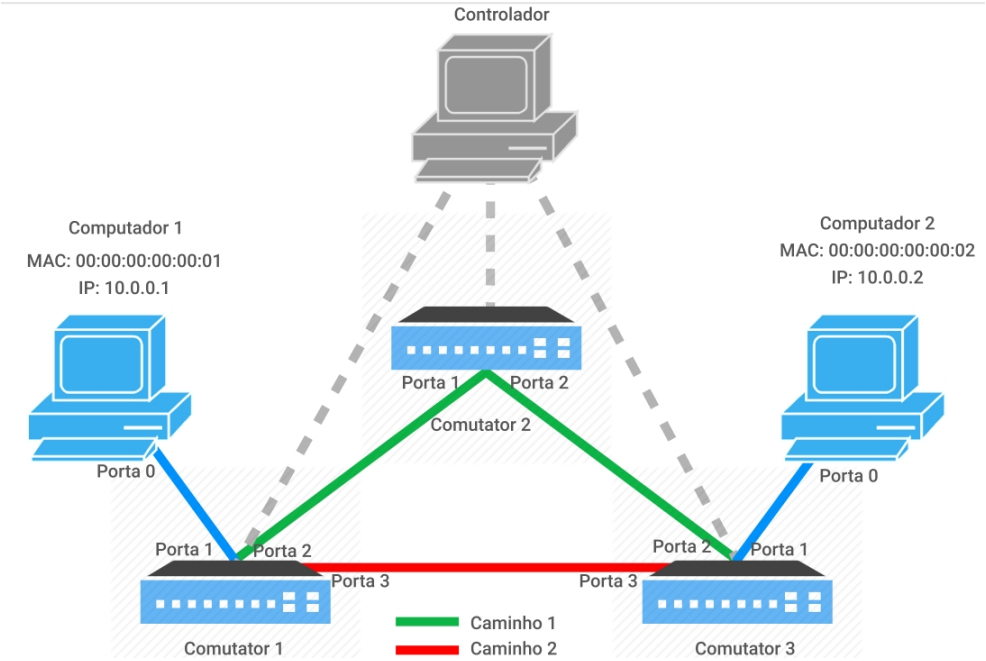
Diferente dos equipamentos de redes tradicionais que faz roteamento baseado em IP, os dispositivos SDN fazem o encaminhamento de pacotes baseados em regras de fluxos. Os equipamentos não tomam nenhuma decisão, apenas aplicam ações de regras contidas nas tabelas de fluxos para encaminhar o pacote à porta correta. Dessa forma os dispositivos de SDN podem escolher qual camada de rede (*e.g.*, L2, L3, L4) utilizará para efetuar o encaminhamento, dando maior liberdade para as aplicações.

Conforme mencionado na Seção 2.2 quando um fluxo é ao comutador e este não possui nenhuma regra em sua tabela para realizar o encaminhamento, o comutador envia para o plano de controle um pacote de *packet-in*, o controlador analisa o cabeçalho do pacote para efetuar a configuração dos dispositivos. Com os dados apresentados no cabeçalho do *packet-in* junto a visão completa da topologia que representam o plano de dados, o controlador pode determinar qual caminho deve ser utilizado. Nessa escolha existem diferentes requisitos que podem ser considerados, como: caminho mais curto, maior vazão, menor tráfego e menor latência. Esses requisitos são utilizados em balanceamento de carga para plano de dados com múltiplos caminhos.

Ao determinar quais dispositivos virão compor o caminho do fluxo, o controlador adiciona

ou atualiza regras nas tabelas desses dispositivos. Assim, os próximos pacotes correspondentes ao fluxo não precisarão mais consultar o controlador, pois cada dispositivo que receber o pacote já está devidamente configurado. Dessa forma o encaminhamento será feito para todo novo fluxo adicionado mantendo a utilização da rede sempre otimizada, conforme os algoritmos escolhidos para efetuar o encaminhamento (AKYILDIZ et al., 2014).

Figura 9 – SDN com dois caminhos possíveis.



Fonte: Elaborada pelo autor (2021).

A topologia ilustrada pela Figura 9 contém dois caminhos possíveis entre o computador 1 e o computador 2. Essa topologia é utilizada para explicar o funcionamento das políticas de RR Seção 3.2, CMCR Seção 3.3, CMCP 3.4 e CMT descrito na Seção 3.5.

3.2 ROUND-ROBIN

A estratégia do RR é utilizar o algoritmo *First In, First Out* (FIFO) para fazer o escalonamento entre caminhos que podem levar o pacote da origem até o destino, sendo essa estratégia implementada de forma proativa. O algoritmo faz o balanceamento de novos fluxos criados, fazendo a redistribuição desses fluxos por caminhos alternativos em tempo real, dividindo a sobrecarga dos enlaces. Em sua estratégia mais clássica, toda vez que um pacote de um determinado fluxo gera um *packet-in* no comutador, o controlador fica responsável por encontrar todos os caminhos de origem a destino e enfileirar estes caminhos em ordem, de tal forma que o caminho selecionado é removido do início e adicionado ao fim da fila. Este processo é repetido para cada novo *packet-in* gerado, fazendo com que os servidores ao se comunicarem utilizem sempre múltiplos caminhos.

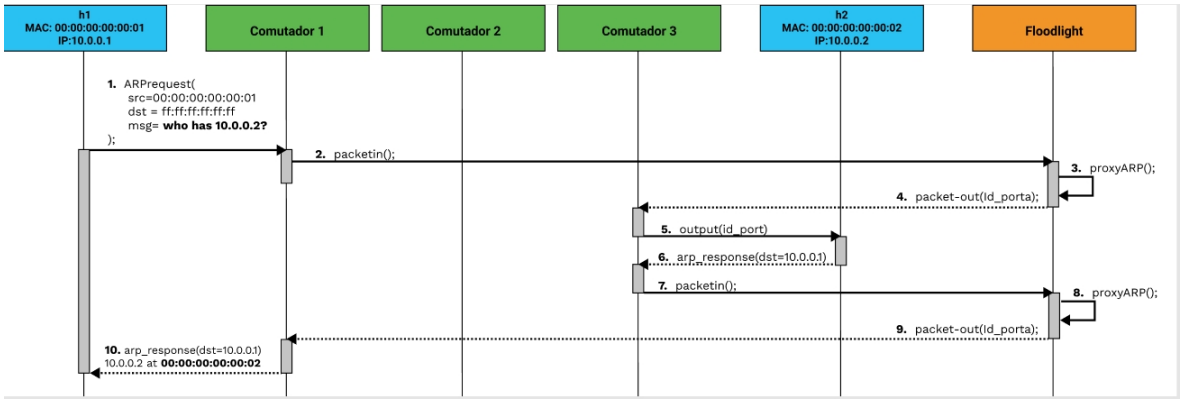
Quando o computador 1 da Figura 9 executar o comando *ping* ao endereço do computador 2 (*e.g*,10.0.0.2), antes de enviar os pacotes de sequência do protocolo ICMP, ele precisa do endereço MAC do destino, para isso é enviado um pacote ARP com MAC destino em *broadcast*. Após a validação do MAC de destino as sequências dos pacotes ICMP começam a ser enviadas. A Figura 10 ilustra o percurso que este ARP realiza na política de RR para a topologia da Figura 9. O evento 1 do diagrama é o envio do ARP *broadcast* com o endereço MAC de destino *FF:FF:FF:FF:FF:FF*. Um *packet-in* é gerado no evento 2 para que o controlador decida a ação. Como o controlador tem a referência do destino é feito um *proxy*, procedimento que entrega o pacote diretamente para o comutador e porta vinculada ao endereço IP do destino, evento 3 e 4, ou possíveis destinos, quando o controlador não tem nenhuma referência conhecida. Nesta topologia, o computador 2 que é o destino, está ligado a porta 1 do comutador 3 . Então o evento 4 do diagrama irá enviar o pacote do *packet-in*, diretamente ao comutador 3 com a ação *output(1)*. O evento 5 ilustra o momento em que o pacote é enviado diretamente ao destino.

Ao receber o pacote ARP que pergunta qual o MAC do endereço 10.0.0.2, o computador 2 é obrigado a responder com a mensagem *10.0.0.2 at: 00:00:00:00:00:02*, porque este é seu endereço, representado na Figura 10 pelo evento 6. Caso o ARP entregue pelo controlador não corresponda o endereço, não seria gerado resposta. A resposta entregue ao comutador 3 não possui regras para tratar o pacote, gerando o evento 7 (*packet-in*). Analisando o cabeçalho do pacote o controlador faz novamente o *proxy* entregando diretamente ao comutador 1 para ser enviado pela porta 1 (evento 9 do diagrama). No evento 10 o comutador aplica a ação que veio junto ao pacote, entregando a resposta ao computador 1. Essa forma de resolução de ARP é a mesma para todas as políticas CMCP e CMT. O controlador não insere regra de fluxo durante o processamento de *packet-in* gerado por pacotes ARP, para que o controlador possa comportar dispositivos móveis conectados a pontos de acesso *wireless* de algumas SDN. Dessa forma, para manter a consistência da rede, após um limite de tempo ocioso, a referência que o controlador possui do dispositivo terminal é removida. Nesse caso em que o controlador não tem essa referência, o pacote é enviado para todas as portas que não são enlaces do tipo comutador-comutador, com intuito de encontrar o dispositivo terminal, ou seja, faz uma operação de *proxy*, enviando o pacote para todas as conexões desconhecidas de modo a validar o endereço do computador. Esse formado de resolução de ARP também é útil em redes com múltiplos caminhos, pois entrega os pacotes *broadcast* diretamente as portas de comutadores ligados a possíveis dispositivos terminais, evitando futuros *packet-in* para o mesmo pacote.

Depois que o ARP é resolvido no evento **10** do diagrama ilustrado pela Figura 10, e o computador que está fazendo a requisição possui a validação do endereço de MAC do destino, então é iniciado o envio das sequências de pacotes. Como o controlador não inseriu nenhuma regra nas tabelas de fluxos dos comutadores, será disparado um *packet-in* já no primeiro pacote. Então é iniciado os eventos começados na sequência **11** do diagrama da Figura 11.

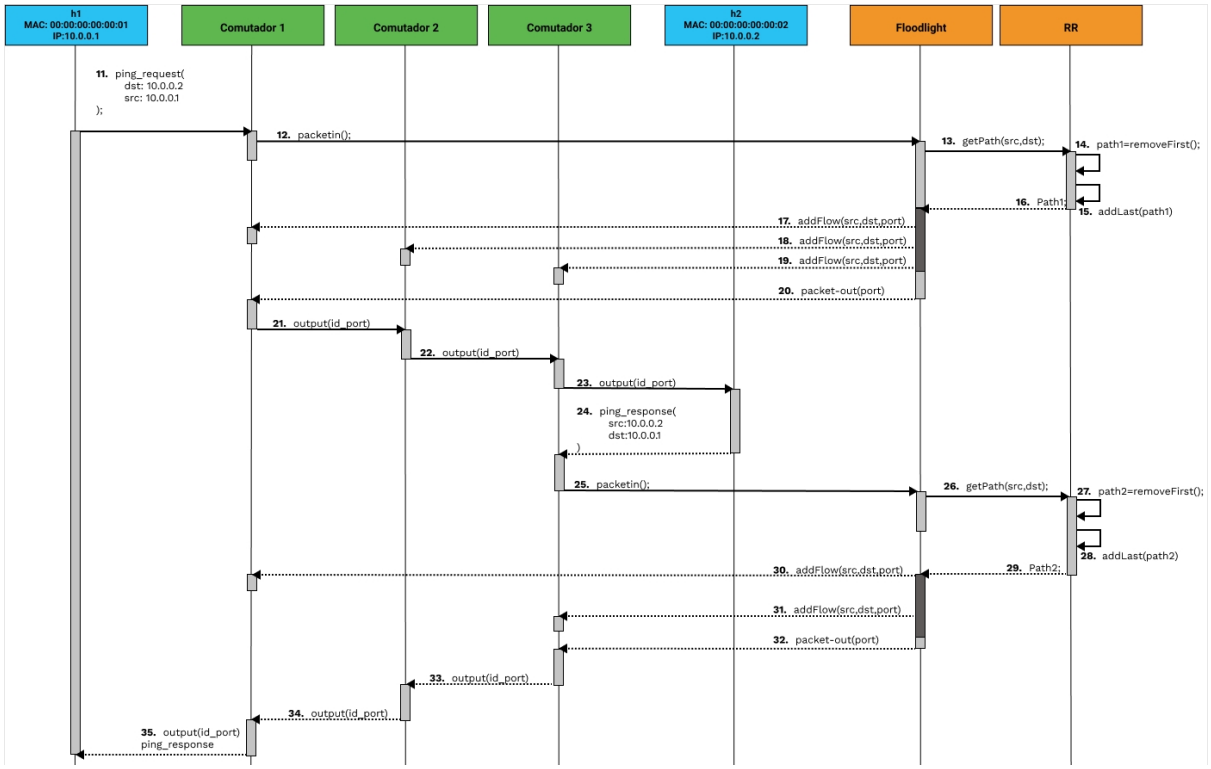
O evento 11 é o envio do primeiro pacote ao comutador. No evento 12 o comutador identifica a falta de *match* mara tratar o pacote, gerando um *packet-in*. O eventro 13 ocorre com

Figura 10 – Diagrama de sequência, funcionamento da resolução de ARP das abordagens Proativas.



Fonte: Elaborada pelo autor (2021).

Figura 11 – Diagrama de sequência, funcionamento da política de *Round-Robin*.



Fonte: Elaborada pelo autor (2021).

o controlador utilizando os endereços contidos no pacote para buscar o caminho entre origem e destino. Neste cenário controlador possui dois caminhos enfileirados em um *cache* interno, caminho 1 e caminho 2 da Figura 9. O evento 14 pega o primeiro caminho nessa fila e o evento 15 passa esse caminho para o fim da fila, a referência do caminho é uma sub lista contendo tuplas de comutador/porta que são necessários configurar para o encaminhamento. Como nesse

cenário o caminho 1 contém três comutadores, é criado um *match* com as informações do pacote e enviado para o comutador 1, 2 e 3 da Figura 9, representados no diagrama pelas sequências 17, 18 e 19. O evento 20 ocorre depois da configuração com o controlador devolvendo o pacote recebido no *packet-in* ao comutador que o gerou. Os eventos 21, 22 e 23 são respectivos ao encaminhamento pela aplicação das regras que determinam a porta de saída. O evento 24 é o momento em que o computador envia a seu comutador a resposta para a sequência 1 do comando *ping*. No evento 25 é gerado outro *packet-in* para tratar a volta do pacote. Passando diretamente ao evento 27, ao selecionar o caminho na fila, o primeiro da lista agora é o caminho 2. O evento 28 passa o caminho 2 ao final da fila e retorna a referência desse caminho contendo o comutador 1 e 3 da Figura 9. O evento 30 e 31 são acionados para inserção das regras e ações nesses dois comutadores. A última ação do controlador é o evento 32, em que o pacote recebido é devolvido ao comutador que o enviou para seguir seu fluxo configurado. A partir deste ponto, os demais pacotes do fluxo são encaminhados conforme a configuração. Se o fluxo não receber pacotes durante 5 segundos, especificado no contador *idl_time* o comutador remove o fluxo.

3.3 CAMINHO MAIS CURTO REATIVO

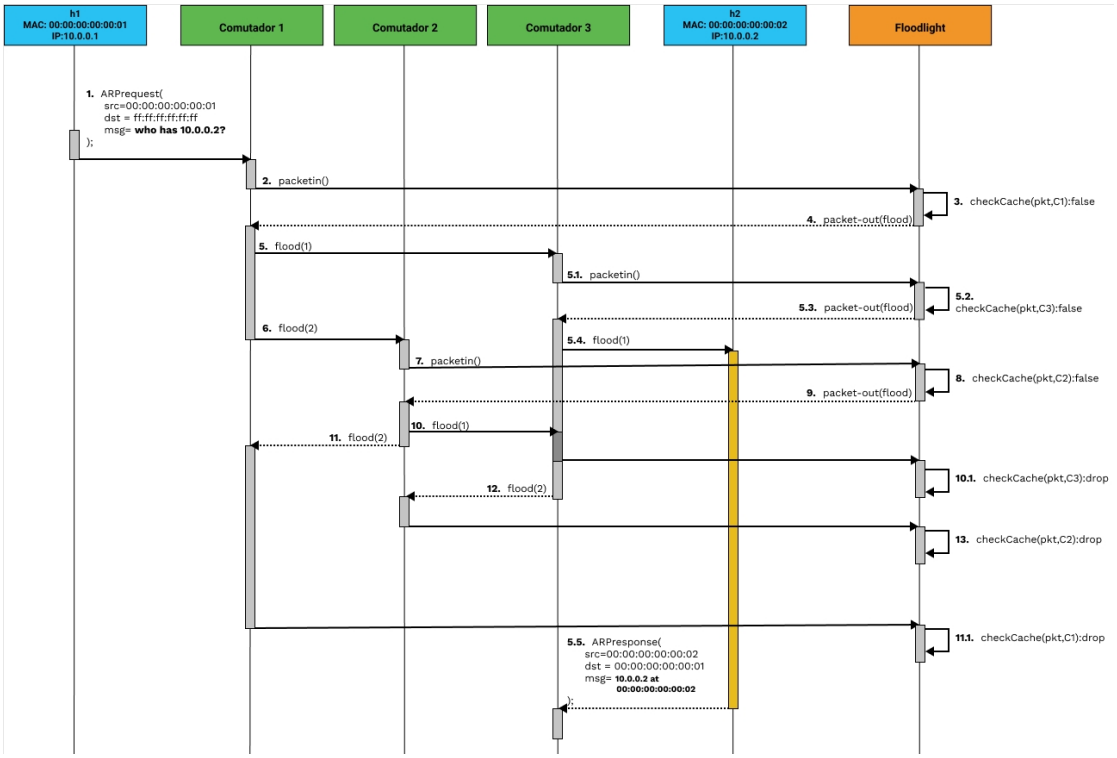
A política CMCR faz o encaminhamento pela rota com menor quantidade de comutadores entre origem e destino reativamente, configurando um comutador por vez, ou seja, quando o controlador recebe o *packet-in* ele configura o comutador que o gerou para que o pacote seja encaminhado para o próximo salto. A resolução do ARP nessa política ocorre conforme o diagrama da Figura 12, já os pacotes gerados após a resolução de ARP são ilustrados pela Figura 13. O cenário utilizado para exemplificar o funcionamento desse algoritmo é o mesmo da Figura 9.

Antes de iniciar o envio de pacotes da sequência *ping*, o computador 1 precisa do endereço MAC de destino. A forma como a política reativa trata os ARPs *broadcast* é muito similar as redes convencionais. Quando o controlador precisa que um comutador realize um *broadcast*, é adicionado a ação *flood* na construção de um *packet-out*. O *flood* é uma especificação interna do *Openflow*, em que o comutador faz uma cópia do pacote para cada porta de um subconjunto de portas permitidas. Ou seja, portas reservadas, bloqueadas e a porta controlador não são utilizadas na execução do *flood*.

O primeiro evento ilustrado na Figura 12 é o envio da solicitação do endereço MAC do destino. O evento 2 é gerado o *packet-in* pela falta de regras. Como o controlador está utilizando a política de controle de fluxo reativa precisa manter um *cache* atualizado que contém informações do comutador e os *packet-in* que ele já enviou, para evitar o afogamento dos enlaces gerados em ciclos no plano de dados. Esse controle é representado no evento 3 do diagrama ilustrado pela Figura 13, na qual, o controlador verifica no *cache* do comutador 1 se este pacote já passou uma vez por ele, como é a primeira vez a verificação é falsa, então o pacote é adicionado ao *cache* com o identificador do comutador 1. Após guardar a referência o controlador insere a regra no

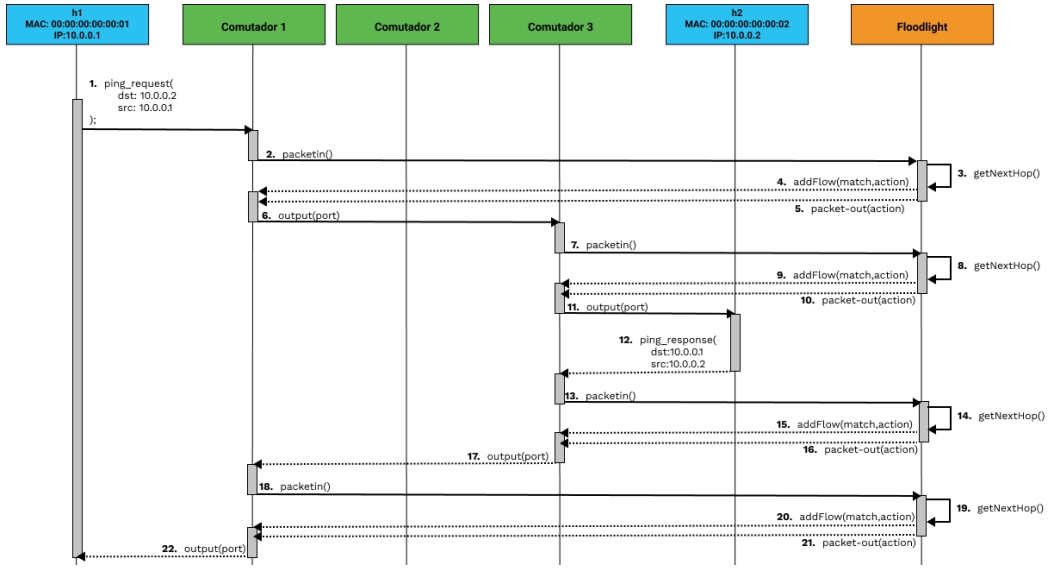
comutador (evento 6) e devolve o pacote recebido pelo *packet-in* com uma ação *flood*.

Figura 12 – Diagrama de sequência, resolução de ARP na política de encaminhamento reativa.



Fonte: Elaborada pelo autor (2021).

Figura 13 – Diagrama de sequência, funcionamento do caminho mais curto na abordagem reativa.



Fonte: Elaborada pelo autor (2021).

Ao processar a instrução *flood* recebida do controlador, o comutador faz a cópia do pacote e o envia para todas as portas válidas, que neste cenário é representada pelos eventos 5 e 6. A partir desse momento o pacote é processado em paralelo pelo comutador 2 e o comutador 3, pois cada um possui uma cópia do pacote. Os eventos 5.1 e 7 são *packet-in* resultante do recebimento dos pacotes gerados pelo *flood*. O evento 5.2 faz uma verificação do pacote no comutador 3, enquanto o evento 8 realizado em paralelo faz a verificação do pacote no comutador 2. Ambos estão sendo processados pela primeira vez em cada comutador, portanto suas referências são armazenadas no *cache* com a identificação de seus respectivos comutadores. Novamente o controlador irá fazer o *flood* desses pacotes, eventos 5.3 e 9. Quando o comutador 3 realiza a operação de *flood*, uma das portas entrega o pacote ao destino, enquanto a outra porta entrega o pacote ao comutador 2, representado na figura pelo evento 12.

Quando o controlador analisar o *packet-in* gerado pelo comutador 2 no evento 13, é verificado no *cache* que este pacote já passou uma vez por este comutador, portanto pode ser descartado. Caso esse controle não seja feito, os ciclos no plano de dados clonariam os pacotes indefinidamente. O pacote entregue ao computador pelo evento 5.4 gera a resposta do evento 5.5, assim todo o processo é novamente iniciado para que o pacote volte para origem da requisição. O *cache* do controlador responsável por armazenar a referência dos *packet-in* já processados por cada comutador, possuem um tamanho fixo e as referências mais antigas são descartadas para evitar uso excessivo de memória.

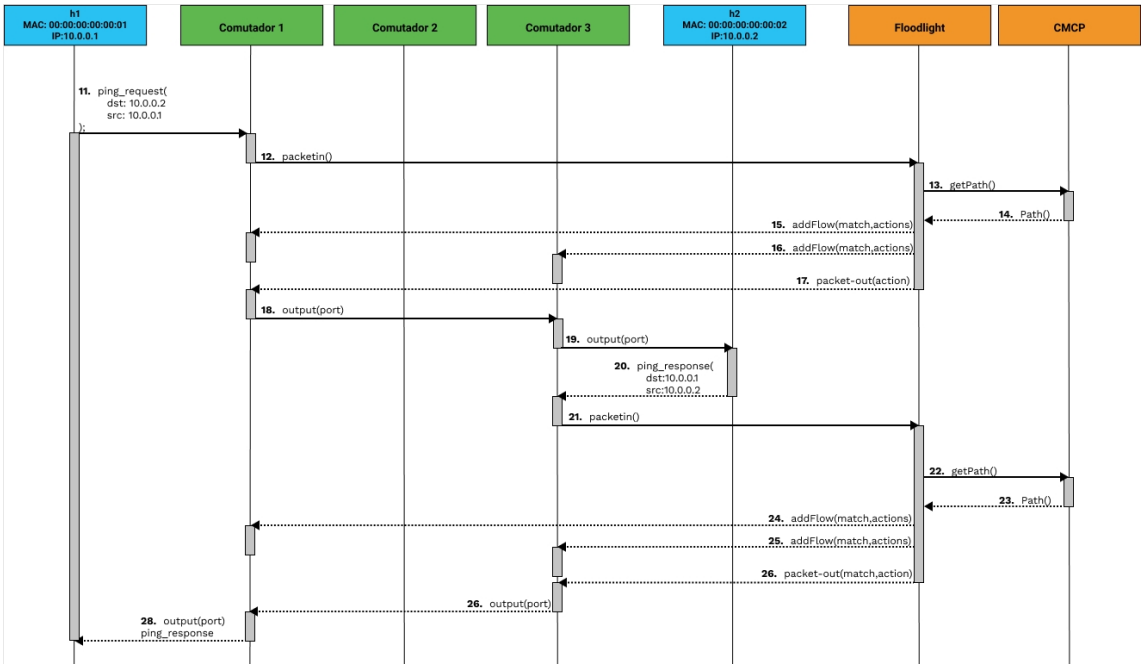
Após os eventos de pacotes ARP o solicitante envia os pacotes do *ping* com os endereços do destino. A figura 13 ilustra o encaminhamento desses pacotes utilizando a política reativa. O evento 1 corresponde ao envio ao comutador, gerando o *packet-in* por falta de regra no evento 2. O controlador busca na topologia interna representada pelo grafo qual o próximo salto do caminho de menor número de saltos, representado pelo evento 3 do diagrama. Como esta política considera apenas um salto, é inserido a regra de fluxo no evento 4 e devolvido o pacote ao comutador no evento 5. O evento 6 representa o encaminhamento para o próximo nó da rede, o comutador 3. Um novo *packet-in* é gerado no evento 7 repetindo as ações de configuração que consiste em encontrar o próximo salto, configurar o comutador e devolver o pacote, evento 8, 9 e 10. No evento 11 a aplicação da regra sobre o pacote faz a entrega diretamente ao destino.

O procedimento realizado para o pacote de origem 10.0.0.2 para 10.0.0.1 segue a mesma lógica anterior, eventos de 13 a 22 correspondem a criação de um novo fluxo para que a resposta volte a origem da solicitação.

3.4 CAMINHO MAIS CURTO PROATIVO

A política CMCP, utiliza a rota com menor quantidade de saltos entre origem e destino, configurando todos os saltos no primeiro *packet-in*. Considerando o mesmo cenário de exemplo dos algoritmos anteriores, Figura 9, mesmo que tenha caminhos alternativos o controlador sempre irá escolher o mesmo caminho para criar o fluxo de ida e o fluxo de volta.

Figura 14 – Diagrama de sequência, funcionamento da política de caminho mais curto da abordagem proativa.



Fonte: Elaborada pelo autor (2021).

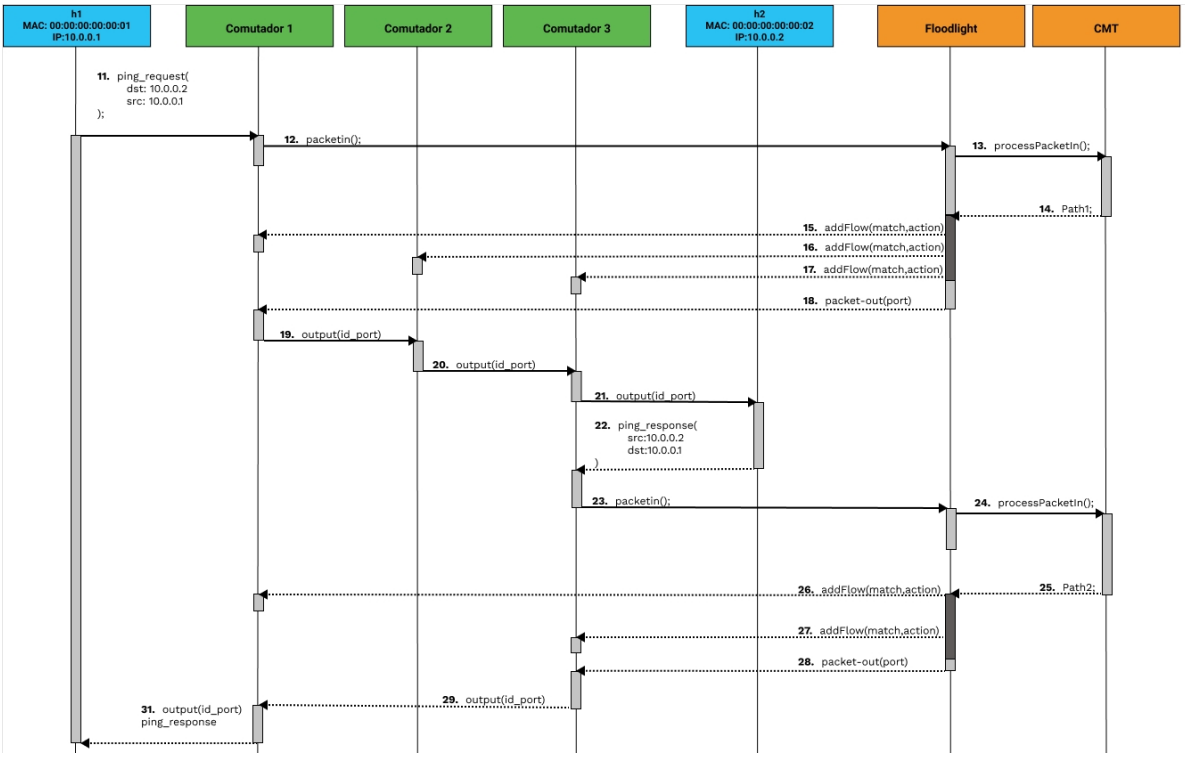
A figura 14 ilustra o funcionamento da política de encaminhamento para o tráfego gerado pelo comando *ping*. Logo apos os eventos de resolução de ARP usando o *proxy* da Figura 10, o próximo evento é iniciado em 11, com a chegada do primeiro pacote da sequência ICMP. Sendo o primeiro pacote o comutador não possui regras em sua tabela para o encaminhamento, gerando o *packet-in* representado na figura pelo evento 12. Ao receber o pacote o controlador faz uma busca no grafo interno que representa a topologia, ilustrado pelo evento 13. Como o grafo tem dois possíveis caminhos, a topologia retorna uma lista ordenada pela quantidade de comutadores em cada caminho, da qual o primeiro da lista é o que possui menor numero de saltos e esta representado pelo evento 14 do diagrama.

Com as informações de comutadores e portas o controlador gera o *match* com ações para cada elemento do caminho e adiciona essas regras nos comutadores, eventos 15 e 16. Após a inserção das regras o controlador devolve o pacote ao comutador que o enviou junto a uma ação de porta de saída, evento 17. A partir deste ponto o pacote segue o fluxo até o destino, evento 18, 19. A resposta gerada pelo destino cria um *packet-in* no evento 21. Como o controlador utilizara novamente o primeiro caminho da lista de caminhos, os comutadores que receberão as regras de fluxo são os mesmos que utilizados anteriormente, eventos 24 e 25. O controlador devolve o pacote no evento 26 para seguir o fluxo, entregando a resposta do *ping* ao solicitante no evento 28. Este algorítimo possui o funcionamento mais simples que os demais, porém utiliza o mesmo caminho mais vezes.

3.5 CAMINHO DE MENOR TRÁFEGO

Os comutadores que utilizam o protocolo *Openflow* fornecem diversas informações sobre os dispositivos. As informações como bits transmitidos e enviados por cada portas podem ser medias periodicamente para se obter estatísticas como taxa de transferência e assim identificar os enlaces que estão sendo menos utilizados. Tal informação permite a criação de algoritmos que calcula o tráfego de todas as rotas existentes entre origem e destino, assim quando o controlador faz a requisição da rota de menor tráfego esta informação já se encontra pronta para uso.

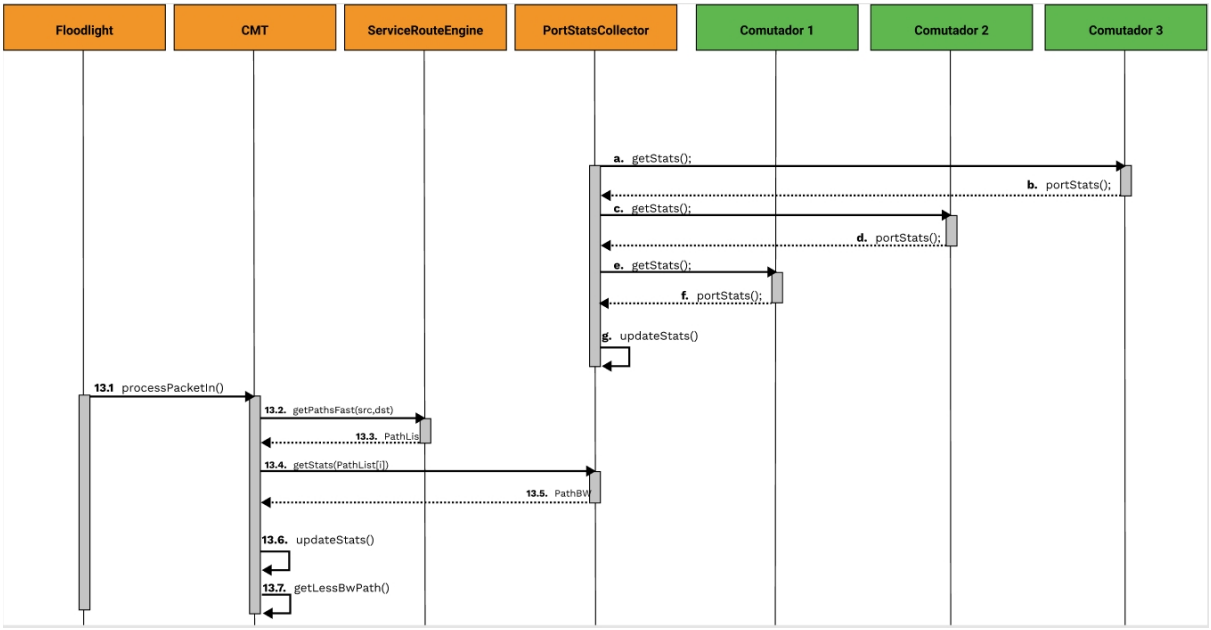
Figura 15 – Diagrama de sequência, funcionamento da política de caminho de menor tráfego.



Fonte: Elaborada pelo autor (2021).

A Figura 15 ilustra o funcionamento do algoritmo para o envio de um pacote entre ***h1*** e ***h2*** com o comando *ping* no plano de dados representado pela Figura 9. Diferente do algoritmo que considera a quantidade de saltos como caminho mais curto, este utiliza a soma de dados transferidos por cada enlaces do caminho para encontrar qual tem o menor trafego. Caso o caminho 1 que possui maior número de saltos apresente menor utilização de enlaces este é selecionado como caminho de menor tráfego. A vazão total do tráfego é limitado pelo enlace entre o comutador e o computador. Ou seja, se esse enlace é de *1 Gb/s* então mesmo que existam rotas alternativas entre os comutadores não é possível que essa velocidade seja superada. O diagrama da Figura 15 possui troca de informações divididas em três cores. Em azul estão os computadores, em cor verde os comutadores e em laranja as classes internas do *Floodlight* que foram expandidas na Figura 16 com maiores detalhes. Os eventos da figura 15 começam na

Figura 16 – Diagrama de sequência, troca de informações internas no algoritmo de caminho de menor tráfego.



Fonte: Elaborada pelo autor (2021).

sequência 11 após a resolução de pacotes ARPs, e terminam na sequência 31, com a resposta do comando *ping*. Os eventos da Figura 16 que vão da sequência 13.1 à 13.7 ocorrem entre os eventos 13 e 14 da Figura 15.

A Tabela 4 possui uma descrição detalhada sobre os principais eventos que ocorrem para a configuração de um novo fluxo conforme a política de menor tráfego. Sendo que os eventos de **a** até **g** são tarefas paralelas executadas periodicamente a cada segundo, esses são responsáveis por manter a taxa de utilização dos enlaces, calculados e armazenados no *cache* interno do controlador para consultas, de modo a minimizar o tempo de resposta ao *packet-in*.

Tabela 4 – Descrição dos eventos usados para configuração do fluxo na política de menor tráfego.

Evento	**Descrição**
11. ping request	O primeiro pacote da sequência *ping* já contendo os endereços resolvidos pelo ARP é enviado para o comutador
12. packetin	Como o comutador não possui regra de match para o cabeçalho do pacote, este é enviado ao controlador
13. processPacketIn	O controlador desenpacota a menssagem na qual são extraidas as informações necessárias
13.2 getPathFast	Busca em um grafo/cache local todos os caminhos possíveis entre origem e destino
13.4 getStats	Para cada caminho existente é buscado a soma do trafego entre os enlaces
13.6 updateStats	A lista é ordenada de forma crecente de tráfego
13.7 getLessBwPath	Seleciona o caminho de menor tráfego, gerando uma lista de tuplas com comutador e porta
15, 16, 17. addFlow	É gerado um match com as informações do pacote, cada elemento da lista de tuplas tem sua respectiva ação relacionada a porta de saída. As regras são então enviadas aos comutadores
18. packet Out	Realizado após a adição dos fluxos no switch, devolve o pacote novamente ao comutador que gerou o packet-in para que ele siga o fluxo configurado
19, 20, 21	Comutadores aplicam as ações relacionada ao match instalado anteriormente
22. ping_response	O computador de destino envia a resposta para o request
23. packetin	Como o controlador configurou apenas a rota de 10.0.0.1 para 10.0.0.2, agora é necessário gerar as regras para tratar pacotes de 10.0.0.2 para 10.0.0.1
31. ping_response	A resposta do *ping* é entregue ao solicitante
Evento	**Descrição**
a,c,e	Controlador envia para cada comutador da rede uma solicitação de estatísticas.
b,d,f	Pacotes enviados dos comutadores contendo todas as estatísticas de portas e entradas da tabela de fluxo.
g	As informações são atualizadas em um cache interno para consulta de consumo de comutador/porta

Fonte: Elaborada pelo autor (2021).

3.6 TRABALHOS RELACIONADOS

Foram encontrados na literatura seis trabalhos relacionados a avaliação e comparação de políticas de controle de fluxo em SDN. A Tabela 5 traz um levantamento sobre as políticas, métricas de avaliação, topologias e ambientes utilizados para comparação dos algoritmos.

Em (AKIN; KORKMAZ, 2019), foi efetuado um estudo de modo a poder comparar a

Tabela 5 – Trabalhos relacionados a comparações de políticas de encaminhamento de fluxo.

Referências	Algoritmos	Métricas	Topologia	Ambiente
(AKIN; KORKMAZ, 2019)	Classe 1: 1. *Minimum Hop Algorithm* (MHA); 2. *Shortest Path* (SP); 3. *Widest-Shortest Path* (WSP). Classe 2: 1. *Dynamic Shortest Path* (DSP); 2. *Dynamic Widest-Shortest Path* (DWSP). Classe 3: 1. *Minimum Interference Routing Algorithm* (MIRA); 2. *Least Interference Optimization Algorithm* (LIOA); 3. *Improved Least Interference Optimization Algorithm* (ILIOA).	1. Total de dados transferidos(*throughput*); 2. Quantidade de pacotes perdidos; 3. Tempo de computação do caminho.	1. *MIRANET*; 2. *ANSNET*;	Plano de controle: RYU; Plano de dados: *Mininet*.
(ICHIKAWA et al., 2013)	1. *Warshall-Floyd* (Proposta); 2. *Shortest Path*;	1. Total de dados transferidos (*Throughput*); 2. Tempo de execução do NAS.	1. Topologia intra-domínio: (EUA, Japão, Singapura, Tailândia)	Plano de controle: Não especificado; Plano de dados: *Open vSwitches*.
(NAEEM; TARIQ; POOR, 2020)	1. *Shortest Path*; 2. *Lagrangian relaxation-based aggregated cost* (LARAC); 3. Sway; 2. SEQOS baseado em *Bellman–Ford*; (Proposta).	1. Custo energético; 2. Total de dados transferidos (*Throughput*).	1. *Goodnet*; 2. *AttMpls*.	Plano de controle: POX; Plano de dados: *Mininet*.
(CASAS-VELASCO; RENDON; FONSECA, 2020)	1. *Reinforcement Learning and Software-Defined Networking Intelligent Routing* (RSIR); 2. Shortest Path (Dijkstra): 2.1. Tráfego; 2.2. Taxa de perda de pacotes; 2.3. Latência; 2.4. Composto (Latencia + Perda + Vazão).	1. Total de dados transferido (Throughput);	1. GÉANT: 23 comutadores.	Plano de controle: Ryu; Plano de dados: Mininet.
(MARCONDES et al., 2016)	1. Shortest Path; 2. Random Selection; 3. Round-Robin over Multiple Paths;	1. Tempo de execução do NAS.	1. Fat Tree: 8 servidores e 10 comutadores.	Plano de controle: Floodlight; Plano de dados: Mininet.
(ZHANG; YAN, 2015)	1. Open Short Path First (OSPF) - redes tradicionais; 2. Shortest Path - SDN.	1. Tempo de resposta das requisições.	1. Topologia de 16 comutadores; 2. Topologia de 120 comutadores.	1. Rede tradicional: 1.1. Controle: Quagga; 1.2. Mininext. 2. SDN 2.1. Plano de controle: Floodlight; 2.2. Plano de dados: Mininet.

Fonte: Elaborada pelo autor (2021).

eficiência entre três classes de algoritmos para controle de fluxo, algoritmos de encaminhamento com: 1) custo de enlace estático, 2) custo do enlace dinâmico e 3) custo do enlace dinâmico com interferência mínima. Na primeira classe contém tês algoritmos de encaminhamento que consideram custo estático como quantidade de saltos. A segunda classe possui duas variações para o caminho mais curto dinâmico, em que é utilizado largura de banda para calcular os pesos de enlaces. A terceira classe contém três algoritmos que adicionam pesos a enlaces criticos junto aos pesos de custo dinâmico. O trabalho comparou essas classes e concluiu que algorítimos dinâmicos tem um desempenho melhor que os estáticos, mas não possuem vantagens significativas entre si dentro de sua própria classe. Em relação à coleta de estatísticas da rede para os algoritmos de custo dinâmico, os autores destacaram duas abordagens. Uma busca manter a projeção atualizada da camada de dados no controlador de modo que o controlador tenha informações o suficiente para identificar os caminhos mais utilizados e estimar largura de banda. A outra abordagem para coleta de informações é fazer leituras periódicas dos comutadores. Esta última abordagem foi utilizada na política de CMT apresentada na Seção 3.5, e os desafios encontrados por este forma leitura são descritos em maiores detalhes na Seção 5.9.

Em (ICHIKAWA et al., 2013) o autor propõe um agente para monitorar informações de latência e vazão permitindo o controlador aplicar no algoritmo de *Warshall-Floyd* e encontrar o melhor caminho. Para validar a proposta o autor utilizou o programa *Numerical Aerodynamic Simulation* (NAS) descrito em detalhes na Seção 4.2. O resultado obtido pelo tempo de execução das aplicações foram comparados com os resultados obtidos pela política de menor caminho e concluiu que a solução proposta teve um desempenho melhor.

Em (NAEEM; TARIQ; POOR, 2020), o autor faz um estudo comparativo entre políticas de controle de fluxo em relação à eficiência do gasto de energia para *Industrial Internet of Things* (IIoT). O autor considera para um bom gerenciamento do tráfego a perda de pacotes, *jitter* e latência em relação ao QoS. O trabalho está relacionado a engenharia de tráfego e demostra como uma decisão tomada no plano de controle pode resolver das mais diferentes categorias de problemas apenas na forma como efetua o gerenciamento. Os resultados do autor mostra que é possível atingir uma redução de até 50% do custo energético violando apenas 5% do *Service Level Agreement* (SLA), resultado aceitável quando se trata de IIoT.

Ainda relacionado a alternativas de algoritmos de encaminhamento, o trabalho de (CASAS-VELASCO; RENDON; FONSECA, 2020) utiliza roteamento com aprendizagem por reforço como alternativa a gerenciamento do controle de fluxo. A proposta faz leituras periódicas de estatísticas do plano de dados para alimentar um novo plano acima do plano de gerenciamento, chamada *Knowledge Plane*, que extrai e analisa esses dados gerando o caminho ótimo, na qual, é considerado tanto a largura de banda, perda de pacotes e latência durante a avaliação dos caminhos. Para comparação da proposta o autor utilizou variações do algoritmo de *Dijkstra* [1], cada uma dessas variações foram baseadas em latência, perda de pacotes, largura de banda e ainda uma versão composta entre latência e largura de banda. A proposta apresentou melhor eficiência ao comparado com o *Dijkstra* latência e *Dijkstra* perda de pacotes, porém teve 3% a menos de eficiência em relação ao *Dijkstra* com largura de banda.

Em (MARCONDES et al., 2016), o autor compara a política de *Shortest Path* com as políticas de *Random Selection* que escolhe o caminho de forma aleatória, e a política de *Round-Robin over Multiple Paths* cujo funcionamento é baseado em fila circular para escalonamento de caminhos, apresentado na Seção 3.2. O autor cria uma topologia *Fat Tree* com 8 servidores e 10 comutadores utilizando o *mininet*. Para comparar as políticas utilizou o *framework* NAS e com o tempo de execução das aplicações foi possível identificar que a utilização do *Shortest Path* induz a criação de gargalos na camada de dados. Dessa forma o algoritmo *Round-Robin over Multiple Paths* teve melhores resultados. O desempenho dos algoritmos de múltiplos caminhos são discutidos com mais detalhes na discussão dos resultados Seção 5.9.

Em (ZHANG; YAN, 2015), o autor realiza um estudo entre o protocolo utilizado em redes tradicionais intra-domínio *Open Short Path First* (OSPF) e a política de encaminhamento padrão do controlador *Floodlight*. Para o estudo o autor criou um ambiente com o *Mininet* e o *Floodlight* emulando uma topologia simples com 16 comutadores e outra topologia mais complexa com 120 comutadores. Para gerar as mesmas topologias na forma de redes tradicionais o autor utilizou o emulador *Mininext*, versão adaptada do *Mininet* para fazer roteamento baseado em *IP* utilizando o protocolo OSPF do *framework* oferecida pelo *Quagga* que é um pacote de programas de roteamento que contém os principais protocolos de roteamento de redes tradicionais. O trabalho utiliza o tempo de resposta gerado pela ferramenta *httping* para comparar os dois cenários. Na qual, conclui que o algoritmo de redes tradicionais possui uma vantagem sobre o de SDN quando

[1] Dijkstra é um algoritmo eficiente para encontrar o caminho mais curto em um grafo com pesos em aresta.

o trafego é reduzido, porém, conforme o volume de dados e a escala da topologia é aumentado o protocolo de encaminhamento de fluxo do SDN possui maior vantagem em relação ao tempo de resposta das requisições. O protocolo OSPF utiliza o caminho com menos quantidades de saltos durante o roteamento, esse caminho é calculado durante a inicialização da rede, e seu funcionamento é similar ao funcionamento da política CMCP, assim seu comportamento em relação ao aumento do tráfico também é notável durante a análise dos resultados na Seção 5.

3.7 CONSIDERAÇÕES PARCIAIS

Neste capítulo foi explicado o funcionamento dos algoritmos de controle e encaminhamento de fluxos comumente utilizados em SDN. Dentre políticas existentes citadas na Seção 3.6 para tratar o encaminhamento de fluxos, foram selecionados apenas duas abordagens baseadas em caminho mais curto, e duas abordagens baseadas em balanceamento de carga para redes de múltiplos caminhos. Nas de caminho mais curto, cujo objetivo é encontrar a rota com menor quantidade de saltos, uma foi elaborada de forma proativa (CMCP) e outra reativa (CMCR). Para políticas que utilizam balanceamento de tráfego, foi implementado o (RR), que faz o escalonamento entre os caminhos alternativos através de listas circulares. A segunda política de múltiplos caminhos foi (CMT) que usa as informações de *bits* transmitidos por porta oferecidas pelo protocolo *OpenFlow* para determinar o caminho de menor trafego. O Capítulo 4 discute a elaboração de uma rede SDN emulada para o estudo do comportamento desses algoritmos apresentados.

4 PROTOCOLO EXPERIMENTAL

O preço de comutadores SDN baseados em *OpenFLow* estão se tornando cada vez mais baratos com o passar dos anos, porém ainda possuem um custo elevado para elaboração de estudos em redes de larga escala, de modo que os emuladores ainda são os mais utilizados em experimentos, principalmente no meio acadêmico. O presente Capítulo tem o objetivo de apresentar a ferramenta gratuitas para criação e avaliação de cenários. A Seção 4.1, apresenta um emulador que permite a criação de SDN baseadas no protocolo *OpenFLow*. A Seção 4.2 apresenta uma aplicação baseada na comunicação MPI para comparação das políticas. A Seção 4.3 tem o objetivo de apresentar uma topologia de múltiplos caminhos, utilizada como cenário de teste. Por fim a Seção 4.4 estabelece a métrica necessária para o estudo.

4.1 EMULADOR MININET

Mininet é uma emulador que permite criar uma rede virtual executando em um único computador ou MV através de comandos básicos para prototipação de redes tradicionais e SDN em larga escala. A ferramenta permite interagir de maneira fácil com a infraestrutura SDN por meio de uma interface de linha de comando (*Command-line interface* (CLI)) e APIs, tornando-o frequentemente utilizado no desenvolvimento de pesquisas (MININET, 2016).

Por permitir a criação de uma infraestrutura com comutadores *OpenFlow*, o *Mininet* é uma ferramenta acessível para realização de casos de uso e comparação dos algoritmos de controle de fluxos. Por padrão o emulador possui quatro topologias básicas para realização de testes podendo ser facilmente alterada através de adição e remoção de servidores, comutadores, controladores e enlaces, que podem instanciados e adicionados a infraestrutura. Por se tratar de um emulador o controlador *Floodlight* pode obter o controle de gerenciamento da rede virtual como se esta fosse um SDN físico, permitindo assim realizar as comparações das políticas de encaminhamento.

4.1.1 Arquitetura

Os principais componentes arquiteturais do *Mininet* são detalhados a seguir (LANTZ; HELLER; MCKEOWN, 2010).

- *Enlaces*: Cria um par virtual que conecta um comutador a um computador permitindo assim o envio de pacotes de uma interface a outra. A classe *TCLink* permite que seja definidos largura de banda e latência para os enlaces.

- *Computador:* No *Mininet* um computador é um processo *shell* no qual cada instancia pode possuir um processo isolado, com sua própria interface *eth0*, sendo que a comunicação é isolada e só pode ser realizada entre esses processos pela de rede através dos enlaces.

- *Comutadores*: Os comutadores *OpenFlow* do *Mininet* oferecem a mesma semântica de encaminhamento de pacotes oferecido por um comutador *OpenFlow* real, permitindo o gerenciamento por um controlador externo aderente ao padrão.

- *Controlador*: O controlador pode ser executado tanto na rede virtual quanto externamente em um computador separado atreves de uma rede física.

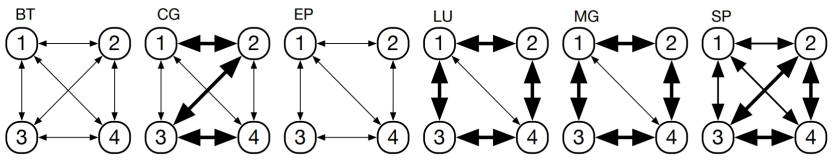
Para auxiliar a criação de uma infraestrutura, o *mininet* fornece uma API desenvolvida em *Python*, e cada componente pode ser facilmente instanciado e adicionado a rede organizados de forma lógica para representar a uma rede real.

4.2 APLICAÇÃO PARA TESTES

Comparar e avaliar políticas de controle de fluxo é uma tarefa em que se considera o volume de tráfego transmitidos em um determinado intervalo de tempo. Levando em conta que a infraestrutura analisada não é uma infraestrutura física real, existem diversas barreiras que devem ser consideradas, tais como o escalonamento de processos do sistema operacional, infraestrutura, controlador e os processos utilizados para realizar troca de mensagens, que podem influenciar nos resultados.

Para essa avaliação foi utilizado o programa *Numerical Aerodynamic Simulation* (NAS) (PETERSON; JR; BAILEY, 1984), ferramenta que contém um conjunto de programas desenvolvidos pela *National Aeronautics and Space Administration* (NASA), criados especificamente para avaliações de supercomputadores paralelos em diversos tamanhos de redes e *grid* computacionais. Construído na linguagem de programação *C*, esta ferramenta utiliza *Message Passing Interface* (MPI) na construção de aplicações paralelas que trocam informações entre os servidores para a resolução de tarefas, sendo elas *Block-Tridiagonal* (BT), *Conjugate gradient* (CG), *Embarrassingly parallel* (EP), *Integer sort* (IS), *Lower Upper* (LU), *Multigrid* (MG) e *Solution Pentadiagonal* (SP).

Figura 17 – Padrão de comunicação da aplicação NAS em SDN.



Fonte: Elaborada por (MARCONDES et al., 2016).

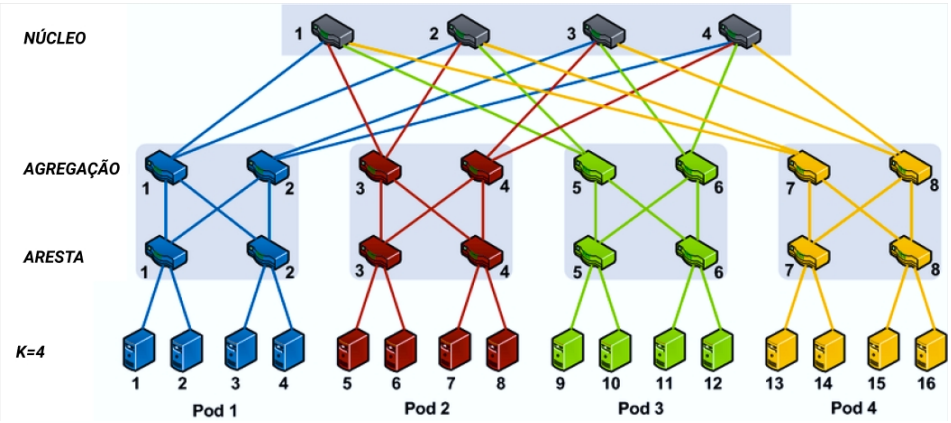
A figura 17 ilustra o padrão de comunicação entre os processos separados em 4 servidores, na resolução de suas tarefas (MARCONDES et al., 2016). A ilustração é um grafo em que cada vértice representa um servidor e as arestas representam o volume de dados trafegados entre os servidores. O volume de dados é representado pela largura das arestas, assim arestas finas

representam pouca comunicação enquanto as mais largas representam maior comunicação. Nesse ambiente, ordenando as aplicações por volume de dados trafegados, EP necessita a troca de 10 KB para resolução de sua tarefa, a segunda menor aplicação BT necessita transmitir 48.04 MB, a terceira aplicação MG transfere 101.24 MB, CG transfere 137.23 MB, a aplicação LU 496.80 MB e a de maior volume de transferência é SP com 1,707.55 MB. O aumento da quantidade de servidores faz com que se aumente também a troca de informações devido à divisão da tarefa entre os processos MPI, porém as informações servem como base para se entender a diferença de desempenho das políticas em relação ao tempo de resoluções dos problemas.

4.3 TOPOLOGIA

A topologia escolhida para a análise dos algoritmos foi a *Fat Tree*. Por se tratar de uma topologia comumente utilizada em alguns *datacenters* contendo rotas alternativas e fornecendo maior escalabilidade. Criada com o auxílio da API *python* do emulador *Mininet*, essa topologia possui 3 camadas, a primeira chamada de núcleo, ligada com a segunda camada de agregação. A terceira camada possui os comutadores de arestas que são ligadas aos comutadores de agregação e os computadores da rede. (ALQAHTANI; HAMDAOUI, 2018).

Figura 18 – Design de uma *Fat Tree* com *4 POD*.



Fonte: Elaborada pelo autor (2021), baseado em (AL-FARES; LOUKISSAS; VAHDAT, 2008).

Em uma *Fat Tree*, *k* representa a quantidade de PODs em que cada um deles possuem duas camadas de *k/2* comutadores por camada. Cada *k* quantidade de portas de um comutador da camada inferior (aresta) é dividida entre *k/2* computadores e o restante das *k/2* portas conectadas a camada superior de agregação. A primeira camada é o núcleo e contém $(k/2)^2$ comutadores sendo que cada uma das *k* portas são conectadas a um POD diferente. Com isso, o número de POD é uma especificação que determina a relação entre portas e quantidade de comutadores. Por exemplo, com k=4, é obtido uma *Fat Tree* com 4 POD, possuindo 4 comutadores por POD, em que cada comutador possui 4 portas divididas em dois *uplinks* e 2 *downlinks*. Ou seja, cada comutador de aresta possui 2 computadores em seus *downlinks*, e dois comutadores de

agregação em seus *uplinks*. Os comutadores de núcleos possuem as 4 portas como *downlinks* ligadas uma em cada POD (AL-FARES; LOUKISSAS; VAHDAT, 2008). A Figura 18 ilustra essa configuração de uma *fattree* com 4 POD, dando um total de 16 computadores na rede.

Essa topologia foi emulada em um computador com 4 GB de RAM e um processador intel core i3. Os enlaces são de *1 GB/s* e 4 PODs foi o número máximo de maior estabilidade nas execuções dos programas do NAS. Cada um dos 16 servidores da *Fat Tree* executam um *daemond SSH* de modo a suportar as aplicações MPIs.

4.4 MÉTRICA PARA ANÁLISE

Avaliar o desempenho de políticas de encaminhamento envolvendo diversos servidores é uma tarefa que pode considerar métricas como vazão, latência, perda de pacotes e tempo de resposta das requisições. Por exemplo, se for avaliar o *download* de um servidor a outro, a eficiência dependeria do tráfego já existente na rede, e para isso seria necessário obter o mesmo tráfego para as diferentes políticas para que nenhuma seja prejudicada. Portanto, para facilitar foram definidas métricas em relação ao tempo que uma aplicação distribuída em vários servidores leva para solucionar um problema imutável. Assim ao executar essa mesma aplicação em diferentes políticas é encontrado tempos de execuções conforme a eficiência dos algoritmos de encaminhamento de fluxo. Para isso o trabalho utilizou de várias aplicações do *benchmark* NAS, cuja comunicação entre os servidores é crucial para a conclusão da tarefa. Ao término das execuções é obtido através de arquivos de *log* o tempo que a tarefa levou para concluir. Sendo assim, foram feitas 10 execuções de cada aplicação do *framework* para cada política apresentada. Os resultados obtidos foram agrupados para geração dos gráficos discutidos no Capítulo 5.

4.5 CONSIDERAÇÕES PARCIAIS

Este Capítulo fez uma revisão sobre o ambiente utilizado para comparação das políticas de encaminhamento de fluxo. Fez também a introdução a ferramenta *Mininet*, usada para emular uma *Fat Tree OpenFlow* e introduziu a aplicação NAS, que por resolver tarefas que necessitam comunicações entre servidores tem sido utilizada em pesquisas acadêmicas, para avaliar computadores de alto desempenho em nuvens e *clusters*. O Capítulo 5 tem o intuito de fazer um estudo sobre os resultados obtidos pelas aplicações do NAS e comparar a eficiência das políticas implementadas.

5 ANÁLISE DOS RESULTADOS

O presente Capítulo tem o objetivo de fazer a análise dos resultados obtidos por cada aplicação do *benchmark* disponibilizado pelo NAS nas comparações das políticas de encaminhamento de fluxo. A seção 5.1 contém o necessário para a análise dos gráficos em relação às aplicações executadas. As demais Seções de 5.2 até 5.8 fazem a apresentação dos resultados individuais obtidos por cada aplicação do NAS. A Seção 5.10 discute as diferenças e similaridades existentes nas políticas segundo os resultados obtidos.

5.1 APRESENTAÇÃO DOS DADOS

Os gráficos das seções 5.2 a 5.8 são representados da seguinte forma: cada gráfico é um programa do *NAS* que executou um *slots* MPI por computador da *Fat Tree*, totalizando 16 *slots*, ou seja, 16 processos. Foram feitas 10 execuções de cada aplicação do NAS por política, dessa forma foi obtido o tempo de execução para elaboração dos gráficos das Seções 5.2 à 5.8. O eixo vertical representa o tempo em segundos que o programa levou para concluir a tarefa, este eixo foi deslocado da origem para facilitar a observação. No eixo horizontal estão as políticas de encaminhamento de fluxo.

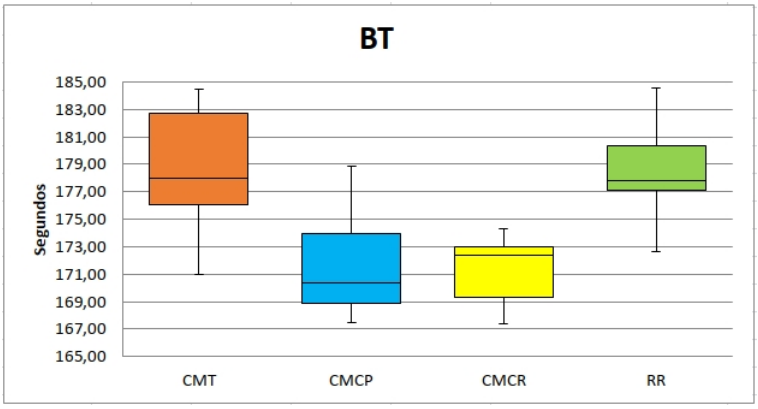
O CMT em laranja é a política que considera o consumo da largura de banda para encontrar o caminho de menor tráfego explicado na seção 3.5. Em azul, CMCP é a política de encaminhamento proativa que utiliza o caminho mais curto detalhado na Seção 3.4. Em amarelo os intervalos de tempo levados pela aplicação com a política CMCR, que utiliza o caminho mais curto reativamente explicado na Seção 3.3. Por último, o RR em verde é a política de *round-robin* detalhado na Seção 3.2.

Cada elemento do gráfico em ordem crescente contém: valor mínimo representado pelo início do traço vertical inferior; primeiro quartil (25% da amostra); segundo quartil representa os 50% (mediana); terceiro quartil (75% da amostra) e o fim do traço superior representa o maior tempo realizado. Os retângulos coloridos apresentam 50% do conjunto de dados entre o primeiro e o terceiro quartil. Os intervalos entre mínimos e primeiro quartil contém 25% da amostra assim como nos intervalos entre o terceiro quartil e o máximo.

5.2 *BLOCK-TRIDIAGONAL*

A aplicação BT, é um algoritmo de *Gauss* para sistemas tridiagonal de blocos. O gráfico ilustrado na Figura 19 representa a execução do programa para as políticas apresentadas. Começando pela política CMT, o menor tempo obtido foi de 171 segundos com um primeiro quartil de 176,11 s e uma mediana de 178,02 s. O terceiro quartil representando o 75% da amostra foi de 182,76 s enquanto o maior tempo obtido por esta política foi 184,48 s. A política CMCP teve o menor tempo em 167,51 s com um primeiro quartil de 168,86 s, mediana de 170,41 s e intervalo

Figura 19 – Tempo de execução da aplicação BT.



Fonte: Elaborada pelo autor (2021).

de 50% dos valores da amostra fechou em 173,94 s no terceiro quartil. O maior tempo obtido por esta política foi 178,89 s.

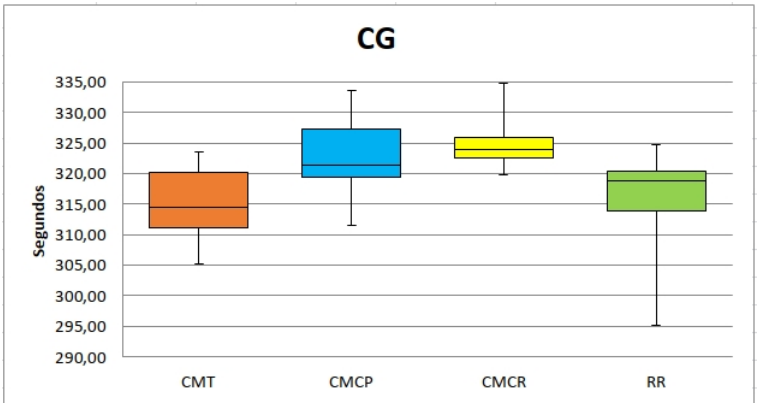
A política de CMCR teve o menor tempo em 167 segundos como a política proativa. O tempo do primeiro quartil foi de 169 s, mediana de 172,35 s e terceiro quartil de 173,03 s. O intervalo de 50% e o terceiro quartil apresentaram pouca variação de tempo sendo o maior tempo de execução dessa política foi de 174,34 s. A política de RR teve o menor tempo atingido em 172,64 s, com o primeiro quartil de 177,10 s, mediana de 177,86 s e terceiro quartil de 180,39 s. O maior tempo obtido nessa política foi 184,53 s.

Para os testes da aplicação BT, a política de CMCP apresentou a menor mediana de tempo seguido pela sua variante reativa CMCR, as políticas que atingiram maiores tempos são as que utilizam caminhos alternativos. A aplicação BT cria um total de 1070 fluxos transmitindo aproximadamente 48 MB durante seu processamento, a baixa comunicação faz com que as políticas que consideram o menor caminho CMCP e CMCR tenham melhores desempenhos por selecionarem caminhos com menos saltos, já que a quantidade de tráfego não é suficiente para gerar atrasos nos enlaces.

5.3 *CONJUGATE GRADIENT*

A aplicação de Gradiente Conjugado (CG) possui alta taxa de transferência de pacotes entre os 16 servidores, o tempo de execução é ilustrado pela Figura 20. A política CMT teve o menor valor de tempo obtido em 305,17 s com o primeiro quartil de 311,10 s, mediana de 314,45 s e terceiro quartil aproximado de 320,28 s. O maior valor obtido nas 10 execuções da aplicação BT para esta política foi de 323,56 s. Para a política CMCP o menor valor de tempo foi de 311,53 s, primeiro quartil de 319,44 s, mediana de 321,48 s e seu terceiro quartil de 327,28 s. O maior valor tempo atingido foi de 333,61 segundos. A política reativa CMCR apresentou o menor valor de 319,90 s com um primeiro quartil de 322,59 s, sua mediana foi

Figura 20 – Tempo de execução da aplicação CG.



Fonte: Elaborada pelo autor (2021).

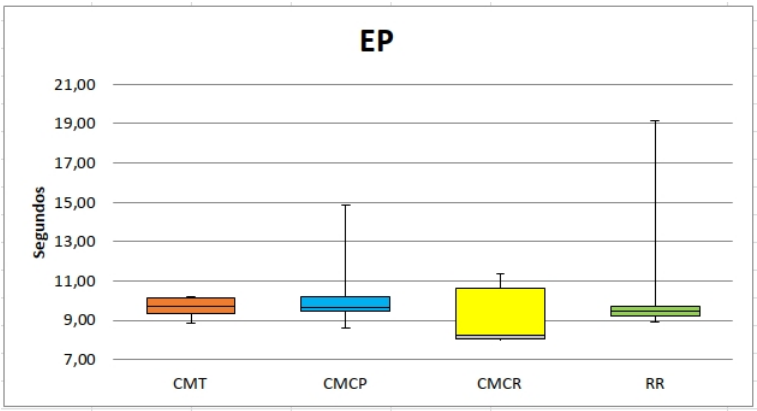
de 323,88 s com um terceiro quartil de 325,83 s. O maior tempo que essa política levou para concluir BT foi 334,75 s. A política de RR apresentou o tempo mínimo em 295,07 s e primeiro quartil de 313,93 s, mediana de 318,79 s e o terceiro quartil de 320,48 s. O maior resultado de tempo obtido foi 324,75 s.

Dentre as políticas CMT obteve a menor mediana (314,45 s) seguido pela política RR, ambas apresentaram tempos menores em relação a variantes do caminho mais curto CMCP e CMCR. A aplicação CG cria 812 fluxos pelos quais são trafegados 101 MB, devido à alta taxa de transferência é possível notar uma queda na vantagem dos caminhos mais curtos com o aumento de tráfego. Em (ZHANG; YAN, 2015), em que o autor compara o desempenho do protocolo OSPF com a política fluxo CMCP de SDN, este comportamento também é observado com o aumento de fluxos, sendo que OSPF tem o desempenho melhor ao CMCP com pouco tráfego, e pior ao CMCP com o aumento do tráfego.

5.4 *EMBARRASSINGLY PARALLEL*

A aplicação de teste EP é um programa paralelo com pouca troca de informação entre os processos. Ilustrado pela Figura 21 necessita transmitir aproximadamente 10 KB para a realização da tarefa (MARCONDES et al., 2016). Executando essa aplicação com a política CMT foi obtido um mínimo de 8,84 s, primeiro quartil de 9,33 s e uma mediana de 9,70 s. A variação de tempo nesta política foi mínima com o terceiro quartil fechando os 50% do valor da amostra em 10,13 s e um máximo de 10,22 s. Para política de CMCP o mínimo obtido foi de 8,62 s, primeiro quartil e a mediana possuíram valores bem próximos sendo o primeiro de 9,46 s enquanto a mediana 9,66 s. O terceiro quartil foi de 10,20 s e o maior valor obtido de 14,88 s. A política de CMCR apresentou um mínimo de 7,95 s, primeiro quartil 8,04 s, mediana de 8,22 s e o terceiro quartil de 10,61 s. O máximo obtido foi de 11,35 s. Para a política de RR foi obtido um mínimo de 8,93 s, primeiro quartil que representa 25% dos valores apresentou

Figura 21 – Tempo de execução da aplicação EP.



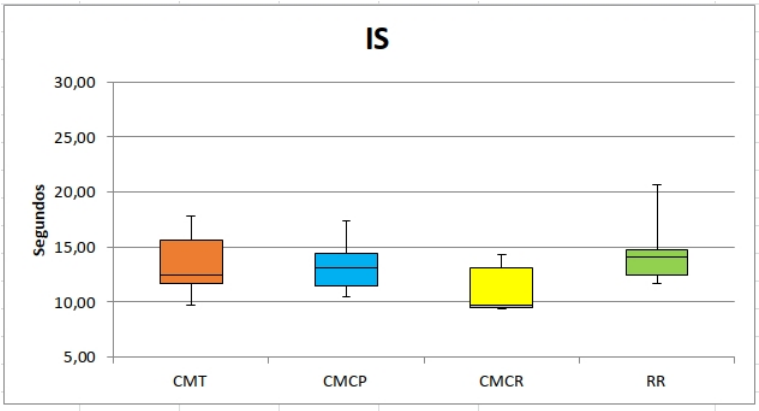
Fonte: Elaborada pelo autor (2021).

9,21 s e mediana de 9,49 s. O terceiro quartil representando os 75% dos valores foi de 9,69 s e o maior tempo de 19,15 s, sendo o que atingiu o maior máximo.

A aplicação EP cria 714 fluxos pelos quais são transferidos 10 KB durante seu tempo de execução, a baixa variação de tempo representado no gráfico sugere que com pouca informação trafegada não há diferença visível entre as políticas, ainda assim, o esperado seria de que as políticas de caminho mais curto CMCP e CMCR tivessem maior vantagem, conforme mencionado na Seção 5.2 e 5.3. A variação de RR ocorre devido à escolha de rotas com muitas quantidades de saltos, é possível notar que os 75% da amostra dessa política tiveram variações entre 9 e 10 segundos.

5.5 *INTEGER SORT*

Figura 22 – Tempo de execução da aplicação IS.

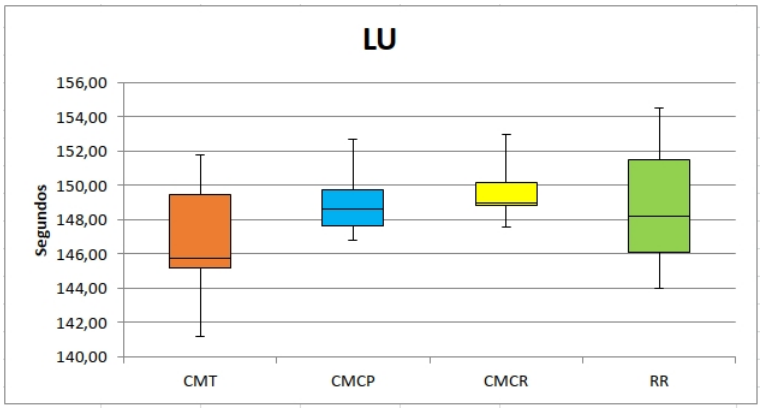


Fonte: Elaborada pelo autor (2021).

O programa IS assim como BT e EP fazem pouca troca de informação em relação as demais, durante sua execução são criados 2384 fluxos, o objetivo desse programa é fazer a ordenação de números inteiros de forma paralela. Representando pela Figura 22, o menor valor obtido pela política CMT é 9,75 s, primeiro quartil de 11,73 s e mediana de 12,41 s. O terceiro quartil é de 15,65 s e o valor de tempo máximo obtido para esta política foi de 17,83 s. Para política CMCP o tempo mínimo foi de 10,45 s, primeiro quartil de 11,45 s e mediana de 13,11 s, o terceiro quartil foi de 14,38 s e o maior tempo obtido nesta política foi de 17,37 segundos. A política de CMCR teve o valor mínimo de 9,43 s bem próximo ao primeiro quartil de 9,46 s e mediana de 9,69 s. O terceiro quartil foi de 13,17 s e 14,32 s. A política de RR teve seu menor tempo de execução em 11,65 s. Seu primeiro quartil 12,47 s e mediana em 14,07 s fechando o intervalo de 75% dos valores com o terceiro quartil em 14,80 s. O maior tempo de execução obtido por esta política na realização da tarefa foi 20,62 segundos. Observando o gráfico é possível perceber que não existe diferença entre as políticas em relação ao desempenho da aplicação IS.

5.6 *LOWER UPPER*

Figura 23 – Tempo de execução da aplicação LU.



Fonte: Elaborada pelo autor (2021).

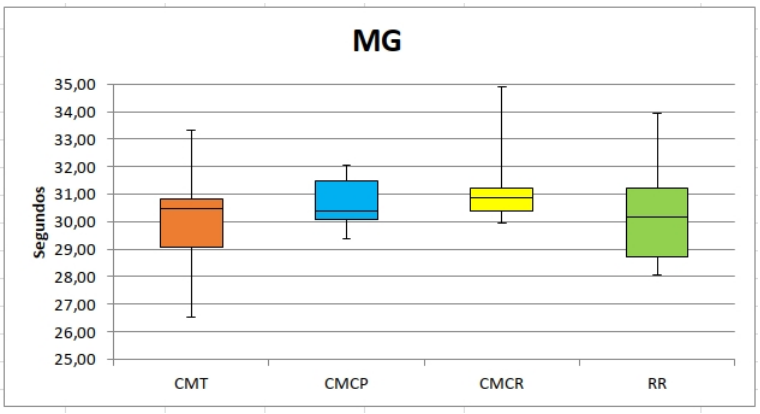
A Figura 23 representa os tempos te execução da aplicação *Lower-Upper Gauss-Seidel solver* (LU). A política CMT teve o menor tempo de 141,18 s, primeiro quartil de 145,17 s próximo da mediana de 145,72 s. O terceiro quartil de 149,45 s e o máximo obtido nessa política foi de 151,78 s. Para a política de CMCP o mínimo foi de 146,81 s e o primeiro quartil de 147,68 s, a mediana foi de 148,61 s e terceiro quartil representando os 75% dos valores foi de 149,76 s. O máximo atingido por essa política de 152,72 s. Para a política de CMCR o menor tempo foi de 147,57 s. O primeiro quartil de 148,85 s, mediana de 148,94 s e terceiro quartil de 150,17 s. O maior tempo atingido foi de 152,95 s. A política de RR teve o menor tempo de

143,98 s, primeiro quartil de 146,09 s e mediana de 148,18 s. O terceiro quartil foi de 151,48 s, o maior tempo obtido de 154,54 s.

A menor mediana foi da política de CMT, que também apresentou o menor tempo de execução. Em 820 fluxos criados foram transferidos entre os servidores 496,80 MB durante o tempo de execução, as políticas de rotas alternativas CMT e RR apresentaram tempos menores que os algoritmos de caminho mais curto. Conforme mencionado em 5.3 o aumento de tráfego dos enlaces começam a gerar atrasos nas políticas que utilizam o caminho mais curto.

5.7 *MULTIGRID*

Figura 24 – Tempo de execução da aplicação MG.



Fonte: Elaborada pelo autor (2021).

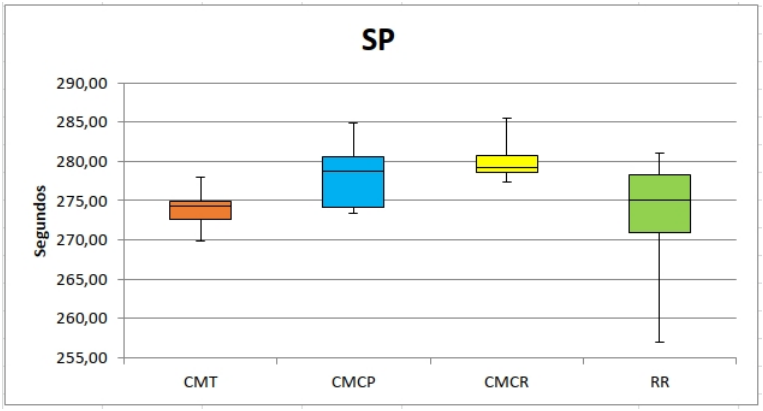
A aplicação *Multi-Grid* (MG) possui um baixo tempo de execução com altas taxas de transferência de pacotes entre os serviços hospedados nos 16 servidores (MARCONDES et al., 2016). Para o teste do MG ilustrado na Figura 24, o menor tempo obtido pela política CMT foi 26,55 s. O primeiro quartil foi de 29,09 s e a mediana de 30,46 s próxima ao valor do terceiro quartil de 30,80 s. O maior valor obtido no intervalo de 10 execuções foi de 33,32 s. A política CMCP teve o menor valor obtido em 29,36 s, primeiro quartil de 30,06 s, mediana de 30,37 s e terceiro quartil de 31,46 s. O maior tempo de execução dessa política foi de 32,03 s. A política de CMCR teve seu menor tempo de 29,93 s, primeiro quartil de 30,38 s, mediana de 30,85 s e terceiro quartil de 31,22 s. O maior tempo no intervalo de 10 execuções foi de 34,89 s sendo também o maior tempo global de execução para esta aplicação. A política RR teve o menor tempo de 28,05 s, primeiro quartil de 28,73 s, mediana de 30,17 s e terceiro quartil de 31,23 s. O maior tempo obtido nessa política foi de 33,95 segundos.

A aplicação MG tem um volume de dados trafegados na rede de 101.24 MB durante seus 30 segundos de execução, na qual são criados 838 fluxos. Observando o gráfico é possível notar que todas as políticas tiveram a mesma mediana (30 s). A troca de informações não é alta o suficiente para gerar gargalos na rede mas também não apresenta vantagens para rotas

alternativas embora estas tenham conseguido obter 25% dos valores de tempos menores que as CMCP e CMCR.

5.8 *SOLUTION PENTADIAGONAL*

Figura 25 – Tempo de execução da aplicação SP.



Fonte: Elaborada pelo autor (2021).

A aplicação SP do *benchmark* NAS tem o maior volume de dados transferido durante a solução da tarefa, assim como uma das maiores taxas de transferências dentre todas as aplicações. A Figura 25 representa o tempo de execução dessa aplicação em relação das políticas. Para CMT o menor tempo foi de 269,78 s com o primeiro quartil de 272,59 s, mediana de 274,29 s e o terceiro quartil de 274,88 s. O maior tempo levado por essa aplicação foi 277,95 segundos. A política CMCP teve o menor tempo de 273,40 s, primeiro quartil de 274,16 s, mediana de 278,81 s e terceiro quartil de 280,61 s. O maior tempo foi de 284,92 s. A política de CMCR teve seu menor tempo em 277,36 s, primeiro quartil de 278,53 s, mediana de 279,26 s e terceiro quartil 280,76 s. O seu maior tempo foi de 285,50 s. Já a política de RR teve seu menor tempo de 256,91 s apresentando também o menor mínimo global. O primeiro quartil foi de 270,87 s, mediana de 275,06 s, terceiro quartil de 278,29 s e maior tempo obtido por esta política foi de 280,99 segundos.

Os tempos de execução apresentou a menor mediana com as políticas de CMT e RR em aproximadamente 174 segundos. A política que teve o menor mínimo apresentado foi o RR com 256 segundos. Os que tiveram maior máxima foram o CMCP e o CMCR, próximos a 180 segundos. Conforme as análises anteriores, já era esperado que para essa aplicação SP tivesse vantagem as políticas de múltiplos caminhos devido ao auto volume de dados transferidos, foram criados 1760 fluxos pelos quais foram transferidos um total de (1707 MB). Sendo assim, quanto maior é a transferência mais visível é a diferença entre essas políticas.

5.9 DISCUSSÃO DOS RESULTADOS

Em relação análise dos gráficos é possível perceber que aplicações com pouco volume de tráfego possui uma vantagem nas políticas que utilizam o caminho mais curto CMCP e CMCR (saltos), como é o caso das aplicações BT, EP e IS. Isso ocorre por não haver tráfego suficiente para causar sobrecarga ou atrasos nos enlaces, sendo assim, quanto menos saltos, menor a quantidade de comutadores processando tráfego, assim como escalonamento do processador.

Em algumas aplicações do teste como SP, MG e CG que necessitam maiores trocas de informações, as políticas que utilizam rotas alternativas obtiveram menores mínimos e a variação de 50% da amostra representado ganho na mediana tempo de execução para políticas com rotas alternativas RR e CMT. Por exemplo, na aplicação CG foi apontado pela mediana, cerca de 5 segundos de ganho em relação às políticas CMCP e CMCR ilustrado pela Figura 20. Os resultados demonstram que aplicações com pouco tráfego são favorecidas pesas políticas com menor quantidade de saltos mesmo que utilizem sempre as mesmas rotas, já nas aplicações com maiores tráfegos são favorecidas as políticas de rotas alternativas. Já entre as aplicações que ficam entre baixa e alta troca de informações os resultados são muito próximos entre si, não sendo visível a diferença.

Em relação à política CMT, o menor intervalo de tempo permitido pelo controlador na coleta de bits transmitidos necessários para os cálculos de vazão, são períodos de 1 segundo, o que pode ser considerado um tempo grande para medir fluxos que alternam de forma tão rápida. Autores que utilizaram dessa abordagem na construção de algorítimos com peso dos enlaces dinâmicos como é o caso de (AKIN; KORKMAZ, 2019) também encontraram desafios relacionados a consultas do plano de dados. Quando o controlador faz muitas requisições ao plano de dados, para obter estatísticas mais atualizadas é gerado excesso de tráfego na rede, mas se o controlador acessa em períodos maiores, a informação se tornam desatualizadas. Neste caso é considerado pesos que não refletem exatamente o estado atual da rede. A ainda soluções que propõem manter uma imagem do plano de dados no plano de controle. Assim quando um fluxo é criado ou removido pelo controlador é armazenado essa informação em um grafo. Dessa forma a quantidade de tráfego de um caminho é estimado pela quantidade de fluxos ativos que ele possui. Apesar desse método não necessitar consultar o plano de dados, a estimativa precisa não é uma tarefa fácil. Em (SINGH et al., 2017), o autor propõe uma forma de estimar a largura de banda utilizada pelos fluxos através leituras periódicas, de modo a aproximar a imagem que o controlador tem do plano de dados.

A política RR foi a que apresentou maior dispersão de tempo, como mostra as Figuras 20, 21 e 25. Essa dispersão ocorre devido à alternância de rotas considerar caminhos com muitos saltos desnecessários, assim como a possibilidade de escolha de caminhos que podem conter o tráfego elevado.

5.10 CONSIDERAÇÕES PARCIAIS

O presente Capítulo apresentou os resultados obtidos pelo *benchmark* NAS, na quais são baseadas em resolução de problemas reais em sistemas de processamento paralelo. O Capítulo expressou em forma de gráficos *bloxpot* as aplicações da ferramenta individualmente para cada política implementada, com o objetivo de realizar um estudo sobre o comportamento dessas aplicações em relação da forma como o encaminhamento de fluxo é feito.

É possível notar que embora políticas que utilizam a menor quantidade de saltos para o encaminhamento sejam melhores com baixo volume de tráfego, essa vantagem vai sendo perdida a medida que o volume de tráfego vai aumentando, até que as rotas alternativas começam apresentar vantagem. Como é um ambiente emulado a quantidade de comutadores existentes no caminho influenciam na transferência de dados devido ao número de processos ativos, (*e.g.*, controlador multi-thread, comutadores, ferramenta NAS), essa mudança pode ser visível com o aumento do número de saltos na política de RR causando mais variações.

6 CONCLUSÃO

Com o avanço crescente da tecnologia, o volume de dados trafegados para o consumo de aplicações vem se tornando cada vez maior. Com esse aumento houve a necessidade de infraestruturas mais robustas, tanto por parte de provedores de serviços utilizando nuvens ou infraestruturas de clusters, quanto por parte das infraestruturas consumidoras, como redes de universidades, empresas, órgãos governamentais, entre outros. Considerando essa demanda, as Redes Definidas por Software (SDN) surgiram como uma proposta, que utiliza da estratégia de um plano de controle separado para efetuar tarefas de otimização dos tráfegos, segurança, tolerância a falhas e escalabilidade.

Para a realização do trabalho foi efetuado um estudo geral sobre as SDN no capítulo 2, fazendo uma revisão de literatura sobre a arquitetura, separada em três principais planos: plano de gerenciamento, plano de controle e plano de dados.

No plano de gerenciamento é que se encontra a maioria das aplicações que utilizam os dados fornecidos pelo plano de controle. Controle de acesso, *loadbalancer*, *interface* de usuário são aplicações comumente encontradas neste plano. A revisão do plano de controle, fez um levantamento entre os duas principais categorias de controladores: os distribuídos, criados para dar maior tolerância a falhas e gerenciar redes cuja as infraestruturas são geograficamente distantes; e os centralizados, que o presente trabalho teve maior ênfase com o controlador *floodlight*. Uma camada muito importante do plano de controle é a *interface northbound*, normalmente utilizando o padrão REST esta tem o objetivo de entregar as funcionalidades do controle da rede para ser acessados por URLs, facilitando na diversificação das aplicações (*e.g.*, python, javascript, c#).

O plano de dados é dividido em duas camadas principais, a primeira contém os dispositivos da infraestrutura, comutadores e computadores. Ainda neste plano podemos dar destaque a equipamentos emulados e simulados, que permitem que estudos possam ser executados sem o custo elevado de compras dos equipamentos. A segunda camada do plano de dados é composta pelos protocolos e padrões definidos para haver comunicação com o plano de controle. Nesta camada o trabalho deu ênfase ao protocolo OpenFlow 1.3 *Long-Term Support* (LTS) por ser amplamente utilizado no mercado.

Diferente das redes convencionais, o SDN utiliza fluxos de pacotes. Estes são sequencias de pacotes que correspondem a regras inseridas nos comutadores da rede pelo controlador. Para a tarefa de gerenciamento dos fluxos o trabalho deu ênfase ao controlador *Floodlight*, detalhado na seção 2.3. Em sua arquitetura modular *open source*, desenvolvida na linguagem de programação Java este controlador foi utilizado para adição dos módulos CMCR, CMCP, RR e CMT.

Para criação da infraestrutura foi utilizado no trabalho o emulador *Mininet*. Ferramenta altamente utilizada para modelagem e pesquisas das SDN baseadas no protocolo *OpenFlow*, por ser gratuita permitindo várias instancias executadas em diversos computadores interconectados por uma rede normal. Emulando uma rede SDN bem próxima da realidade.

Com o ambiente gerado foi utilizado para o estudo quatro estratégias distintas para o

controle dos fluxos: *Round-Robin* (RR), caminho mais curto reativo (CMCR), caminho mais curto proativo (CMCP) e caminho de menor tráfego (CMT). A política de RR utiliza a vantagem de múltiplos caminhos para distribuir o tráfego de pacotes em caminhos alternativos utilizando uma fila circular. A política de CMCR utiliza sempre o primeiro caminho de uma lista de menor quantidade de saltos para formar o caminho mais curto, a cada salto o comutador precisa consultar o controlador que decide gerando regras necessárias para que o pacote chegue ao próximo salto, até que todos os comutadores do caminho sejam configurados. A política de CMCP possui essa mesma estratégia da menor quantidade de saltos, porém com a diferença que nessa, o controlador configura previamente todos os comutadores do caminho, reduzindo o número de consultas. A política de CMT usa as informações fornecidas pelo protocolo Openflow de cada comutador para calcular a largura de banda utilizada por cada porta, calculando assim a taxa de tráfego de cada enlace. Com isso permite que o controlador escolha o caminho com menor tráfego para o fluxo.

Os resultados foram obtidos utilizando o NAS, programa de avaliação de desempenho de super computadores e infraestruturas de rede. As aplicações do NAS foram desenvolvidas com MPI e cada uma possui seu próprio padrão de comunicação e processamento, permitindo que o comportamento das políticas possam ser estudadas na utilização de uma aplicação real.

Em suma, os estudos apontam que quando uma aplicação com pouca comunicação interna é executada, as políticas que utilizam o caminho com menor número de saltos se saem melhor, ainda que utilizem sempre os mesmos caminhos, já que estes não geram um tráfego suficiente para ocasionar atrasos na aplicação. As aplicações com maior volume de dados transmitidos (*e.g.*, CG, SP, MG) possuem um desempenho melhor nas políticas RR e CMT vela vantagem dos múltiplos caminhos.

6.1 TRABALHOS FUTUROS

Utilizar os recursos oferecidos pelos dispositivos de encaminhamento da rede para melhorar a gerenciamento do tráfego é uma tarefa que vem sendo explorada ao longo dos anos, a leitura dessas informações ocasionam dois problemas, na primeira: para obter maior precisão o controlador envia pacotes com mais frequência causando sobrecarregando o canal seguro, ou o controlador; no segundo envia com menos frequência e obtém estatísticas com baixa precisão. Em (SINGH et al., 2017), é reforçado a necessidade de aplicações desenvolvidas de modo a permitir uma melhor leitura de estatísticas de fluxos e portas. A aplicação *sFLow* por exemplo, permite com que comutadores *OpenFlow* mandem periodicamente uniformemente em taxas ajustáveis, cabeçalhos de pacote para o monitoramento dos fluxos, de modo que não sobrecarregue o controlador (MOGUL et al., 2010). Gerando eventos a cada 1/1000 pacotes a ferramenta recebe as informações utilizadas para medir a utilização das portas. Outra ferramenta criada para resolver este problema é o *NetFlow*, desenvolvida pela empresa CISCO tem o objetivo de colher as estatísticas sem sobrecarregar o controlador. Com essa perspectiva trabalhos futuros

utilizando uma melhor leitura para a seleção de rotas podem ser usadas para determinar ganhos mais significativos no gerenciamento do plano de dados.

Ainda em relação da otimização do tráfego nas SDNs, pesquisas como *SDPredictNet-A*, utiliza de Redes Neurais Artificiais, treinadas para prever eventos e encontrar o caminho ótimo para os fluxos (SANAGAVARAPU; SRIDHAR, 2021). Em (CASAS-VELASCO; RENDON; FONSECA, 2020), o autor usa aprendizagem por reforço criando um plano acima do plano de gerenciamento. Essas técnicas utilizadas em conjunto com ferramentas eficientes de coleta de estatísticas podem proporcionar uma aproximação mais exata da imagem que o controlador cria em *cache* para representar o plano de dados, podendo proporcionar trabalhos a serem desenvolvidos.

Outras pesquisas relacionadas a engenharia de tráfego, buscam reduzir o consumo de energia. Consolidar os caminhos e desabilitar temporariamente dispositivos comutadores ociosos sem violar o SLA, podem reduzir o custo em SDN de grande porte (TORKZADEH; SOLTANIZADEH; OROUJI, 2021).

REFERÊNCIAS

AKIN, Erdal; KORKMAZ, Turgay. Comparison of routing algorithms with static and dynamic link cost in sdn-extended version. In: **Proceedings of the 16th IEEE Annual Consumer Communications & Networking Conference, Las Vegas, NV, USA**. [S.l.: s.n.], 2019. p. 11–14. Citado 3 vezes nas páginas 45, 46 e 60.

AKYILDIZ, Ian F et al. A roadmap for traffic engineering in sdn-openflow networks. **Computer Networks**, Elsevier, v. 71, p. 1–30, 2014. Citado 2 vezes nas páginas 35 e 36.

AL-FARES, Mohammad; LOUKISSAS, Alexander; VAHDAT, Amin. A scalable, commodity data center network architecture. In: ACM. **ACM SIGCOMM Computer Communication Review**. [S.l.], 2008. v. 38, n. 4, p. 63–74. Citado 2 vezes nas páginas 51 e 52.

ALQAHTANI, Jarallah; HAMDAOUI, Bechir. Rethinking fat-tree topology design for cloud data centers. In: IEEE. **2018 IEEE Global Communications Conference (GLOBECOM)**. [S.l.], 2018. p. 1–6. Citado na página 51.

BENSON, Theophilus; AKELLA, Aditya; MALTZ, David A. Unraveling the complexity of network management. In: **NSDI**. [S.l.: s.n.], 2009. p. 335–348. Citado na página 13.

BERDE, Pankaj et al. Onos: towards an open, distributed sdn os. In: ACM. **Proceedings of the third workshop on Hot topics in software defined networking**. [S.l.], 2014. p. 1–6. Citado na página 22.

BEZERRA, Divanilson Campelo et al. Engenharia de trÁfego em redes definidas por software. **XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)**, 2016. Citado na página 35.

CASAS-VELASCO, Daniela M; RENDON, Oscar Mauricio Caicedo; FONSECA, Nelson LS da. Intelligent routing based on reinforcement learning for software-defined networking. **IEEE Transactions on Network and Service Management**, IEEE, v. 18, n. 1, p. 870–881, 2020. Citado 3 vezes nas páginas 46, 47 e 64.

DORGIVAL, Guedes et al. **Redes Definidas por Software: uma abordagem sistemica para o desenvolvimento de pesquisas em Redes de Computadores**. [S.l.], 2012. Citado 2 vezes nas páginas 23 e 29.

DORIA, Avri et al. **Forwarding and control element separation (ForCES) protocol specification**. [S.l.], 2010. Citado na página 23.

ERICKSON, David. The beacon openflow controller. In: ACM. **Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking**. [S.l.], 2013. p. 13–18. Citado na página 21.

FLOODLIGHT. **Floodlight Is an Open SDN Controller**. 2012. Acessado em: 2015-10-27. Disponível em: <http://www.projectfloodlight.org/floodlight>. Citado 3 vezes nas páginas 20, 21 e 31.

FOSTER, Nate et al. Frenetic: A network programming language. In: ACM. **ACM Sigplan Notices**. [S.l.], 2011. v. 46, n. 9, p. 279–291. Citado na página 20.

GHORBANI, Soudeh; CAESAR, Matthew. Walk the line: consistent network updates with bandwidth guarantees. In: ACM. **Proceedings of the first workshop on Hot topics in software defined networks**. [S.l.], 2012. p. 67–72. Citado na página 19.

GONG, Yili et al. A survey on software defined networking and its applications. **Frontiers of Computer Science**, Springer, v. 9, n. 6, p. 827–845, 2015. Citado na página 24.

GUDE, Natasha et al. Nox: towards an operating system for networks. **ACM SIGCOMM Computer Communication Review**, ACM, v. 38, n. 3, p. 105–110, 2008. Citado 2 vezes nas páginas 20 e 21.

HANDLEY, Mark. Why the internet only just works. **BT Technology Journal**, Springer, v. 24, n. 3, p. 119–129, 2006. Citado na página 16.

ICHIKAWA, Kohei et al. A network performance-aware routing for multisite virtual clusters. In: IEEE. **2013 19th IEEE International Conference on Networks (ICON)**. [S.l.], 2013. p. 1–5. Citado na página 46.

JAFARIAN, Jafar Haadi; AL-SHAER, Ehab; DUAN, Qi. Openflow random host mutation: transparent moving target defense using software defined networking. In: ACM. **Proceedings of the first workshop on Hot topics in software defined networks**. [S.l.], 2012. p. 127–132. Citado na página 19.

JARRAYA, Yosr; MADI, Taous; DEBBABI, Mourad. A survey and a layered taxonomy of software-defined networking. **Communications Surveys & Tutorials, IEEE**, IEEE, v. 16, n. 4, p. 1955–1980, 2014. Citado 2 vezes nas páginas 17 e 19.

JR, Paulo Roberto Vieira; FIORESE, Adriano; KOSLOVSKI, Guilherme. Comparação de políticas para configuração de fluxos em redes definidas por software. In: **Anais do Computer on the Beach**. [S.l.: s.n.], 2016. Citado na página 18.

KIM, Hyojoon; FEAMSTER, Nick. Improving network management with software defined networking. **Communications Magazine, IEEE**, IEEE, v. 51, n. 2, p. 114–119, 2013. Citado na página 13.

KREUTZ, Diego et al. Software-defined networking: A comprehensive survey. **proceedings of the IEEE**, IEEE, v. 103, n. 1, p. 14–76, 2015. Citado 5 vezes nas páginas 13, 14, 16, 17 e 21.

LANTZ, Bob; HELLER, Brandon; MCKEOWN, Nick. A network in a laptop: rapid prototyping for software-defined networks. In: ACM. **Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks**. [S.l.], 2010. p. 19. Citado na página 49.

MARCONDES, Anderson HS et al. Executing distributed applications on sdn-based data center: A study with nas parallel benchmark. In: IEEE. **2016 7th International Conference on the Network of the Future (NOF)**. [S.l.], 2016. p. 1–3. Citado 5 vezes nas páginas 46, 47, 50, 55 e 58.

MCKEOWN, Nick et al. Openflow: enabling innovation in campus networks. **ACM SIGCOMM Computer Communication Review**, ACM, v. 38, n. 2, p. 69–74, 2008. Citado na página 20.

MEDVED, Jan et al. Opendaylight: Towards a model-driven sdn controller architecture. In: IEEE. **2014 IEEE 15th International Symposium on**. [S.l.], 2014. p. 1–6. Citado 2 vezes nas páginas 20 e 22.

MININET. **An Instant Virtual Network on your Laptop (or other PC)**. 2016. Acessado em: 2016-9-21. Disponível em: <http://mininet.org/>. Citado na página 49.

MOGUL, Jeffrey C et al. Devoflow: Cost-effective flow management for high performance enterprise networks. In: **Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks**. [S.l.: s.n.], 2010. p. 1–6. Citado na página 63.

MONSANTO, Christopher et al. A compiler and run-time system for network programming languages. In: ACM. **ACM SIGPLAN Notices**. [S.l.], 2012. v. 47, n. 1, p. 217–230. Citado na página 20.

NAEEM, Faisal; TARIQ, Muhammad; POOR, H Vincent. Sdn-enabled energy-efficient routing optimization framework for industrial internet of things. **IEEE Transactions on Industrial Informatics**, IEEE, v. 17, n. 8, p. 5660–5667, 2020. Citado 2 vezes nas páginas 46 e 47.

NAKAGAWA, Yukihiro; HYOUDOU, Kazuki; SHIMIZU, Takeshi. A management method of ip multicast in overlay networks using openflow. In: ACM. **Proceedings of the first workshop on Hot topics in software defined networks**. [S.l.], 2012. p. 91–96. Citado na página 19.

NUNES, Bruno AA et al. A survey of software-defined networking: Past, present, and future of programmable networks. **Communications Surveys & Tutorials, IEEE**, IEEE, v. 16, n. 3, p. 1617–1634, 2014. Citado 3 vezes nas páginas 16, 21 e 24.

NYGREN, Anders et al. Openflow switch specification version 1.5. 1. **Open Networking Foundation, Tech. Rep**, 2015. Citado 2 vezes nas páginas 25 e 28.

ONF, ONF. **Openflow switch specification version 1.3. 0 (wire protocol 0x04)**. 2012. Citado 3 vezes nas páginas 25, 27 e 30.

PETERSON, VL; JR, WF Ballhaus; BAILEY, FR. Numerical aerodynamic simulation (nas). In: **Large Scale Scientific Computation**. [S.l.]: Elsevier, 1984. p. 215–236. Citado na página 50.

PETTIT, Justin et al. Virtual switching in an era of advanced edges. In: **2nd Workshop on Data Center–Converged and Virtual Ethernet Switching (DC-CAVES)**. [S.l.: s.n.], 2010. Citado na página 23.

PFAFF, Ben; DAVIE, Bruce. The open vswitch database management protocol. 2013. Citado na página 23.

PFAFF, Ben et al. Openflow switch specification, version 1.3. 0. **Open Networking Foundation**, 2012. Citado na página 29.

PHEMIUS, Kévin; BOUET, Mathieu; LEGUAY, Jérémie. Disco: Distributed multi-domain sdn controllers. In: IEEE. **Network Operations and Management Symposium (NOMS), 2014 IEEE**. [S.l.], 2014. p. 1–4. Citado na página 22.

SANAGAVARAPU, Sowmya; SRIDHAR, Sashank. Sdpredictnet-a topology based sdn neural routing framework with traffic prediction analysis. In: IEEE. **2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)**. [S.l.], 2021. p. 0264–0272. Citado na página 64.

SINGH, Manmeet et al. Estimation of end-to-end available bandwidth and link capacity in sdn. In: SPRINGER. **International Conference on Ubiquitous Communications and Network Computing**. [S.l.], 2017. p. 130–141. Citado 2 vezes nas páginas 60 e 63.

SMITH, M et al. Opflex control protocol. **IETF, Apr**, 2014. Citado na página 23.

SONG, Haoyu. Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane. In: ACM. **Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking**. [S.l.], 2013. p. 127–132. Citado na página 23.

TELEGRAPH, Nippon. **Telephone Corporation, RYU network operating system, 2012**. 2018. Acessado em: 2021-8-21. Disponível em: <https://ryu-sdn.org/index.html>. Citado na página 22.

TORKZADEH, Samaneh; SOLTANIZADEH, Hadi; OROUJI, Ali A. Energy-aware routing considering load balancing for sdn: a minimum graph-based ant colony optimization. **Cluster Computing**, Springer, p. 1–20, 2021. Citado na página 64.

XIE, Junjie et al. Control plane of software defined networks: A survey. **Computer communications**, Elsevier, v. 67, p. 1–10, 2015. Citado na página 20.

ZHANG, Hailong; YAN, Jinyao. Performance of sdn routing in comparison with legacy routing protocols. In: IEEE. **2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery**. [S.l.], 2015. p. 491–494. Citado 3 vezes nas páginas 46, 47 e 55.

ZHOU, Wei et al. Rest api design patterns for sdn northbound api. In: IEEE. **Advanced Information Networking and Applications Workshops (WAINA), 2014 28th International Conference on**. [S.l.], 2014. p. 358–365. Citado na página 20.