Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

NoBI: Uma Interface *Northbound* para a Programação Dinâmica de Redes *OpenFlow* com Suporte à Interoperabilidade entre Controladores

César Rocha Vasconcelos

Tese submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - *Campus* I como parte dos requisitos necessários para obtenção do grau de Doutor em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Metodologia e Técnicas da Computação

Reinaldo Cézar de Morais Gomes
Anderson Fabiano Batista Ferreira da Costa
(Orientadores)

Campina Grande, Paraíba, Brasil

V331n Vasconcelos, César Rocha.
NoBI: uma interface Northbound para a programação dinâmica de redes OpenFlow com suporte à interoperabilidade entre controladores / César Rocha Vasconcelos. – Campina Grande, 2018.
160 f. : il. color.

Tese (Doutorado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2018.
"Orientação: Prof. Dr. Reinaldo Cézar de Morais Gomes, Prof. Dr. Anderson Fabiano Batista Ferreira da Costa".
Referências.

1. Redes de Computadores. 2. Interface Northbound. 3. Redes Programáveis. 4. Redes Definidas por Software. 5. Desenvolvimento de API. I. Gomes, Reinaldo Cézar de Morais. II. Costa, Anderson Fabiano Batista Ferreira da. Título.

CDU 004.7(043)

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECÁRIA SEVERINA SUELI DA SILVA OLIVEIRA CRB-15/225

"NoBI: UMA INTERFACE NORTHBOUND PARA A PROGRAMAÇÃO DINÂMICA DE REDES OPENFLOW COM SUPORTE À INTEROPERABILIDADE ENTRE CONTROLADORES"

CESAR ROCHA VASCONCELOS

TESE APROVADA EM 04/06/2018

REINALDO CÉZAR DE MORAIS GOMES, Dr., UFCG
Orientador(a)

ANDERSON FABIANO BATISTA FERREIRA DA COSTA, Dr., IFPB
Orientador(a)

JOSÉ ANTÃO BELTRÃO MOURA, Ph.D, UFCG
Examinador(a)

TIAGO LIMA MASSONI, Dr., UFCG
Examinador(a)

AUGUSTO JOSÉ VENÂNCIO NETO, Dr., UFRN
Examinador(a)

KELVIN LOPES DIAS, Dr., UFPE
Examinador(a)

CAMPINA GRANDE - PB

Resumo

As Redes Definidas por *Software* (SDN, do inglês *Software-Defined Networks*) são um novo paradigma que tem atraído um interesse significativo da academia e indústria de redes de computadores. Ao contrário das redes tradicionais, o paradigma SDN encoraja a separação entre a lógica de controle da rede dos dispositivos adjacentes e introduz a capacidade de orquestrar a rede em alto nível.

Nos últimos anos, esforços de pesquisa crescentes têm endereçado a concepção de interfaces de programação *northbound* (NBI, do inglês *Northbound Interface*), as quais permitem às aplicações de negócio e aos gerentes de redes se comunicarem, apropriadamente, com controladores SDN para programar a rede. Entretanto, apesar da corrente consolidação do protocolo *OpenFlow*, a ausência de uma padronização destas interfaces tem dificultado a criação de código interoperável. Hoje em dia, dezenas de controladores encontram-se disponíveis; porém, cada um é livre para projetar e implementar sua própria NBI. Logo, à medida que o número de controladores SDN tem aumentado dramaticamente, a habilidade dos gerentes de reutilizar funções da NBI tem se tornado uma tarefa problemática e, em alguns casos, impossível, por conta das diferenças entre as linguagens de programação e instruções dos controladores. O resultado é que a ausência de implementações *northbound* interoperáveis frequentemente tem obrigado gerentes a reescrever partes inteiras do código, quando é necessário suportar um novo controlador. Adicionalmente, a migração de conjuntos de instruções de rede existentes para um outro controlador é um processo lento e tedioso, que tipicamente envolve o aprendizado de novas APIs, modelos de dados complexos e várias outras convenções específicas do controlador-alvo.

Para lidar com tal problemática, é apresentada neste trabalho uma interface *northbound* alternativa, denominada NoBI, que permite aos gerentes programar a SDN com facilidade e independentemente do controlador existente. Em particular, a NoBI provê um conjunto de funcionalidades baseado em princípios REST e verdadeiramente interoperável, isto é, pode se comunicar de forma transparente com diferentes controladores. Para satisfazer o requisito de interoperabilidade entre múltiplos controladores, foram desenvolvidos uma nova arquitetura simples de ser instanciada e modelos de dados compatíveis com controladores SDN conhecidos (*Floodlight* e *OpenDaylight*). Em vez de injetar instruções intrincadas e de baixo nível diretamente no código para programar a SDN, gerentes devem interagir apenas com a NoBI. Esta traduzirá todas as requisições de alto nível recebidas em invocações de funções NBI internas e modelos de dados nativos suportados pelo controlador existente, de forma automática.

Demonstrou-se a viabilidade da NoBI em arquiteturas de redes SDN reais contendo diferentes dispositivos *OpenFlow*. Os resultados obtidos mostraram que a NoBI é capaz de programar a SDN em tempo de execução e ainda atender o requisito de interoperabilidade, com um resultado operacional estatisticamente equiparado às implementações estado da arte (*baselines*) disponíveis.

Palavras-chave: Interface *Northbound*, Redes Programáveis, Redes Definidas por *Software*, Desenvolvimento de API.

Abstract

The Software-Defined Networking (SDN) is a new paradigm that is attracting significant interest from both academy and network industry. Unlike tradicional networks, the SDN paradigm enables innovation by encouraging the separation of network's control logic from the underlying devices and introducing the ability to orchestrate the network in a high-level fashion.

In recent years, increasing research efforts have addressed the conception of northbound programming interfaces (NBIs), which allow business-applications and network administrators to properly communicate with SDN controllers and program the network. However, albeit the current consolidation of the OpenFlow, the lack of a standard for the NBI makes it very difficult for administrators to create interoperable code. Currently, dozens of SDN controllers are available, but each one is free to design and implement its own NBI. So as the number of controllers have increased dramatically, the ability of network managers to use NBI functions have become such a problematic and sometimes an impossible task, given the differences in programming languages and instructions among SDN controllers. As a result, the lack of a interoperable NBI implementation often forces administrators to manually rewrite almost all of their code, every time they need to support a new controller. Also, migrating existing network instruction sets to another controller is a tedious and time-consuming process that typically involves learning new APIs, complex data models and various controller-dependent conventions.

To address this limitation, we present an alternative northbound API, called NoBI, that allows administrators to easily program the SDN irrespective of the existing controller. Particularly, NoBI provides a REST-based set of functions that is truly interoperable, and thus may communicate seamlessly with more than one controller. In order to meet multi-controller interoperability requirement, we introduce an new architecture that can be easily instantiated and also new data models compatible with two prominent SDN controllers (Floodlight and OpenDaylight). So rather than injecting intricate low-level instructions directly into the code to program the SDN, administrators will interact only with NoBI. It translates automatically all the high-level requests into internal NBI calls and native data models supported by the underlying controller.

We demonstrated the feasibility of our solution in real single-controlled SDN architectures using different OpenFlow devices. Our results show that our solution is able to program the SDN at runtime and yet fulfill the interoperability requirement, while statistically achieving similar operational results compared to state-of-the-art baselines.

Keywords: Northbound Interfaces, Programmable Networks, Software-Defined Networks, API Development.

Dedicatória

Dedico este trabalho de doutorado aos meus pais amados, Célia e George, com todo o meu amor e gratidão, por tudo o que fizeram por mim ao longo de minha vida, em especial quanto aos esforços empreendidos para oferecer-me educação, bem como a minha esposa, Daniella, com quem eu amo partilhar a vida.

Agradecimentos

A Deus Trino e à Nossa Senhora, pelo amparo e por me ajudarem a alcançar esta conquista.

À minha amada esposa, Daniella, por todo o seu amor, ajuda, companheirismo e paciência despendidos durante toda a longa jornada deste doutorado.

Aos meus orientadores, Reinaldo e Anderson, pela oportunidade de contar com sua amizade, experiência e preciosos direcionamentos durante a condução desta pesquisa. Com vocês, o doutoramento na UFCG foi uma experiência bastante gratificante. Em especial, quero registrar a generosidade do professor Reinaldo, por ter aceitado me acompanhar nesta jornada. Sua dedicação ao trabalho, ética e caráter constituem, verdadeiramente, uma inspiração.

Agradeço a minha irmã, Renata, meu cunhado, Flávio, e sobrinho, Vítor, pela torcida. A presença deste alegre trio sempre renovava minhas energias.

Da mesma forma, agradeço aos colegas do IFPB *Campus* Campina Grande, por terem me permitido afastar das atividades do magistério para dedicar-me, integralmente, à pesquisa científica. Destes, destaco a colaboração fraterna do professor Dr. Marcelo Portela, com quem pude encontrar auxílio na execução de experimentos e ainda palavras de estímulo e de motivação nos momentos difíceis desta pesquisa.

Aos examinadores da banca de proposta de qualificação e de tese, pelas valiosas sugestões e críticas dadas com o objetivo de melhorar a qualidade final deste trabalho. Destaco as constribuições dos professores Tiago Massoni, Antão Moura, Augusto Neto e Kelvin Dias.

Gostaria de tecer um agradecimento especial à professora Francilene, minha orientadora de mestrado, por generosamente redigir e enviar uma carta de recomendação para o processo de seleção de alunos para doutorado do qual participei.

Agradeço aos amigos do LATEC, em especial Petrônio, Saulo, Fellype e Gustavo, bem como ao amigo Ítalo, pelos cafés, pelas conversas descontraídas e por todo o incentivo.

Aos professores do DSC/UFCG, à Paloma, Lyana e à Coordenação da Pós-Graduação, pela atenção e ajuda recebidas.

Conteúdo

Lista de Acrônimos

ANOVA - *Analysis Of Variance*

API - *Application Programming Interface*

ARP - *Address Resolution Protocol*

DoE - *Design of Experiment*

DSCP - *Differentiated Services Code Point*

ForCES - *Forwarding and Control Element Separation*

GENI - *Global Environment for Network Innovations*

GUI - *Graphical User Interface*

HDR - *High Dynamic Range*

HP - *Hewlett-Packard*

HTTP - *Hypertext Transfer Protocol*

IBM - *International Business Machines*

IDC - *International Data Corporation*

IETF - *Internet Engineering Task Force*

IP - *Internet Protocol*

JSON - *JavaScript Notation*

LLDP - *Link-Layer Discovery Protocol*

MMOG - *Massive Multiplayer Online Games*

NEC - *Nippon Electric Company*

NTT - *Nippon Telegraph and Telephone Corporation*

OF-Config - *OpenFlow Management and Configuration Protocol*

ONF - *Open Networking Foundation*

ONOS - *Open Network Operating System*

OVSDB - *Open vSwitch Database Management Protocol*

POF - *Protocol-oblivious Forwarding*

POX - *Python-based Openflow Controller*

QoS - *Quality of Service*

REST - *Representational State Transfer*

RTF - *Real-Time Framework*

SDN - *Software-Defined Networks*

SDK - *Software Development Kit*

SEP - *Static Entry Pusher*

SOAP - *Simple Object Access Protocol*

TCP - *Transmission Control Protocol*

TI - *Tecnologia da Informação*

UDP - *User Datagram Protocol*

VOIP - *Voice Over Internet Protocol*

W3C - *World Wide Web Consortium*

XML - *eXtensible Markup Language*

Lista de Figuras

Lista de Tabelas

Lista de Códigos Fonte

Capítulo 1

Introdução

As redes de computadores interconectados têm revolucionado a forma como a sociedade moderna trabalha, interage e realiza negócios. Hoje em dia, as redes são críticas às interações sociais e comerciais. Não somente é possível trocar dados por meio de uma rede, mas, sobretudo, desenvolver produtos de *software* especificamente projetados para alcançar uma vantagem competitiva sobre sua capacidade.

A evolução das redes convergentes, assim como o surgimento de equipamentos cada vez mais sofisticados, têm levado gerentes de redes a lidarem com um crescente número de tarefas complexas [Kim e Feamster 2013]. Para uma manutenção apropriada de uma rede, faz-se necessário um profundo conhecimento técnico acerca de sua infraestrutura e funcionamento operacional, aliado a um conjunto bem elaborado de políticas de gerenciamento. Frequentemente, para operar os equipamentos que formam a infraestrutura da rede, gerentes usam diferentes interfaces de controle. Estas interfaces tem o papel de oferecer ao gerente um conjunto de abstrações (i.e., um corpo de funções) capaz de configurar e instalar instruções nos equipamentos da rede. Existem equipamentos cujas interfaces de configuração são sistemas *Web* sofisticados, enquanto que outras são baseadas em terminal de linha de comandos. Sob a perspectiva do gerente de uma rede, a instalação e consolidação de instruções de controle por meio destas várias e diferentes interfaces permitem a estes profissionais reagirem ante a eventos que venham degradar o desempenho normal da rede.

Hoje em dia, a elevada complexidade da infraestrutura física das redes atuais, aliada às interfaces de configuração proprietárias (fechadas) e pouco flexíveis de equipamentos, têm dificultado sobremaneira a formulação e instalação de regras para reconfigurar o comporta-

mento da rede e satisfazer determinados requisitos do domínio de negócio. Verifica-se, ainda nos dias de hoje, algumas dificuldades e problemas em relação não apenas à padronização destas interfaces de controle, mas também, de forma não tão abrangente, à aplicação de um mesmo corpo de funções por parte do gerente para operar outras arquiteturas de redes com componentes e características distintas. É neste desafio que este trabalho está focado.

Recentemente, um campo de pesquisa de grande interesse na área de Redes de Computadores tem sido o das redes definidas por *software* (SDN, do inglês *Software-Defined Networks*) [Boucadair e Jacquenet 2014]. Diferentemente de alguns modelos de gerência de redes tradicionais, onde o administrador realiza a configuração manual e individualizada de equipamentos fazendo uso de um conjunto de comandos de baixo nível (e.g., *scripts*), as "redes programáveis" surgem como uma alternativa bastante promissora, por elevar o nível de abstração do processo de elaboração de políticas de gerência e favorecer a instalação destas regras em dispositivos de rede distintos. A arquitetura de uma rede SDN[1] típica é ilustrada na Figura 1.1.

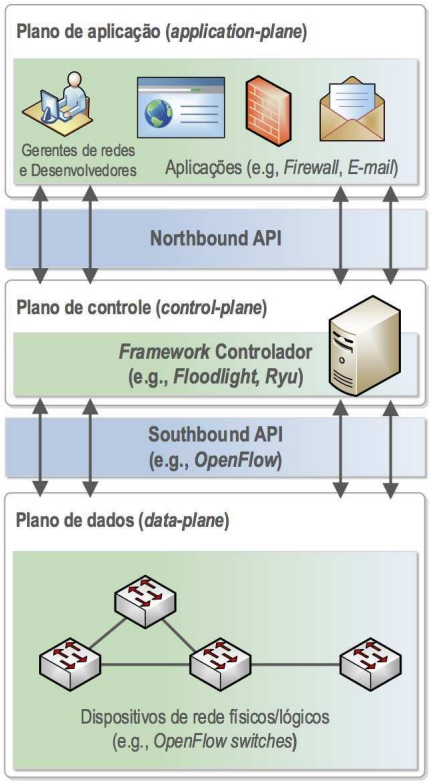


Figura 1.1: Arquitetura típica de uma rede definida por *software*

[1] No restante deste documento, os termos "rede definida por *software* " e "rede SDN" serão usados de maneira intercambiável e com o mesmo propósito.

Na arquitetura SDN, observe que a "inteligência" da rede é logicamente centralizada no chamado plano de controle (do inglês *control-plane*), enquanto que os dispositivos comutadores (i.e., *switches*, roteadores, etc.)—responsáveis *apenas* pela troca de mensagens—compõem o chamado plano de dados (do inglês *data-plane*). A interação entre o plano de controle de uma rede SDN e os vários equipamentos que constituem sua infraestrutura pode ser levada a efeito por meio de um protocolo não-proprietário, como o *OpenFlow* [McKeown *et al.* 2008], por exemplo. Essencialmente, a arquitetura de rede SDN consiste de: a) um plano de dados formado pelos dispositivos de rede comutadores; b) um plano de controle composto de um *framework* controlador (ou sistema operacional de rede), responsável por automaticamente instalar as regras de controle nos dispositivos de rede do plano de dados; e c) o plano de aplicação, usado por gerentes, desenvolvedores e suas aplicações para a definição e repasse de requisitos de controle. Verifica-se, ainda, que a comunicação entre o plano de aplicações de rede e o controlador é levada a efeito por meio de uma interface de programação de aplicações denominada *northbound* API, enquanto que uma API *southbound* permite a interação entre o controlador e os equipamentos da infraestrutura da rede [Haleplidis *et al.* 2015].

Ao observar a Figura 1.1, percebe-se um aspecto inovador das redes SDN: a explícita separação entre os planos de controle e dados. O foco do paradigma SDN é, portanto, "mover" as regras de gerência tradicionalmente instaladas nos dispositivos da rede para o *framework* controlador. Tal realocação potencializa a manutenção, evolução e permite a inovação dos protocolos e da própria rede em si. Estes argumentos são apoiados por vários autores, entre eles, KREUTZ *et al.* [Kreutz *et al.* 2015], que ao abordarem a dicotomia entre o modelo SDN e as arquiteturas de redes tradicionais, citam que o paradigma SDN tem se contraposto à verticalização (i.e., a "inteligência" da rede acoplada e fragmentada em vários dispositivos) das atuais redes de comunicação. Os mesmos autores destacam que a complexidade e rigidez das atuais redes IP e a configuração individualizada de equipamentos devem abrir espaço para a separação entre a lógica de controle e a infraestrutura da rede, a automatização na instauração de políticas de gerência, e a capacidade de programar os equipamentos por meio de estruturas de mais alto nível, independentemente de um fabricante em particular.

Adicionalmente, pode-se afirmar que este recente paradigma traz a reboque outros importantes benefícios, quando equiparado a métodos tradicionais de gerência de redes. Fatores, tais como: (1) a possibilidade de uma abordagem de gerência centralizada, (2) a capacidade

de programar e orquestrar a rede por meio de um *software* controlador, (3) a redução de erros na instauração de políticas de rede, sobretudo quando esta atividade se apoia na instalação manual de comandos em vários equipamentos, e (4) a ênfase no processo de tomada de decisões de mais alto nível, centralizado e amparado por regras de controle que podem ser aplicadas automaticamente e com maior rapidez, apresentam-se bastante encorajadores e apontam o paradigma SDN como um assunto promissor.

A concepção de interfaces de programação *northbound* para ambientes SDN tem sido objeto de interesse de diversos trabalhos encontrados na literatura (vide Capítulo 3). Isto porque tais interfaces podem reduzir a complexidade de implantar instruções de controle nestes tipos de redes. O emprego de estruturas de mais alto nível, exportadas por alguns controladores disponíveis, tem permitido aos gerentes de redes reconfigurar e consultar informações da SDN. Todavia, a ausência de uma padronização mínima destas interfaces faz com que gerentes tenham de lidar com o aprendizado e a variabilidade das funções de programação suportadas pelos controladores, o que aumenta a complexidade e pode causar atrasos para fazer mudanças na SDN. Por exemplo, a Tabela 1.1 mostra exemplos de diferentes formatos de protótipos de funções exportados pelas interfaces de dois controladores SDN conhecidos (*Floodlight* [Big Switch Networks 2013] e *OpenDaylight* [OpenDaylight 2013]).

Tabela 1.1: Exemplos de diferentes protótipos de funções em interfaces de controladores

Controlador	**Formatos dos protótipos de funções**
Floodlight	GET /wm/core/controller/summary/json
OpenDaylight	GET /restconf/operational/network-topology:network-topology/
Floodlight	POST /wm/staticflowpusher/json
OpenDaylight	PUT /restconf/config/opendaylight-inventory:nodes/node/{{switch-id}}/table/0/flow/{{flow-id}}

Primeiramente, na Tabela 1.1, observe que o gerente deve efetuar requisições HTTP usando URIs dependentes do controlador para operar a rede. Especificamente, os dois primeiros formatos de protótipos (linhas 1 e 2) permitem consultar as informações do controlador quanto à topologia da SDN, enquanto que as duas últimas funções permitem ao gerente repassar instruções à rede em um arquivo no formato JSON ou XML, respectivamente. No caso do *Floodlight*, observe que o formato do arquivo a ser enviado (ou recebido) deve ser informado diretamente no final da URI (linhas 1 e 3), o que não é possível no *OpenDaylight*. Além disso, note que o *Floodlight* requer o método HTTP POST para instalar configurações na rede usando o formato JSON, enquanto que o *OpenDaylight* requer uma requisição com o método PUT e o gerente deve fornecer os identificadores do equipamento-alvo e da confi-

guração a ser aplicada na invocação da URI da API. Verifica-se, portanto, que as interfaces *northbound* oferecem abstrações diferentes e que dependem de estruturas, convenções e outras decisões de implementação específicas de cada controlador. O resultado é que gerentes são frequentemente forçados a reescrever instruções de programação da rede existentes quando é necessário aplicá-las em outro controlador distinto. É dentro deste contexto de suporte à programação de redes definidas por *software* que se insere este trabalho de doutorado, tendo em vista a redução do impacto da mudança de controlador sobre o gerente e garantia de interoperabilidade da interface entre controladores distintos.

No restante deste capítulo, é levada a efeito uma discussão da problemática (Seção 1.1) que motivou esta pesquisa, são apresentados os objetivos (Seção 1.2) deste trabalho, sua relevância (Seção 1.3), as principais contribuições (Seção 1.4) desta pesquisa e a organização geral do documento (Seção 1.5).

1.1 Problematização

Ao longo dos anos, vários esforços têm sido empreendidos na especificação de várias interfaces *southbound* para uma interlocução apropriada entre o controlador com o plano de dados SDN, a exemplo dos protocolos ForCES [Yang *et al.* 2004] e POF [Song 2013]. Apesar disso, o protocolo *OpenFlow* [McKeown *et al.* 2008] tem emergido com um padrão *de facto* para mediar a comunicação entre do plano de controle e os inúmeros dispositivos que compõem a infraestrutura física da rede SDN. O *OpenFlow*, portanto, expõe uma interface *southbound* bem definida e compatível com os equipamentos comutadores de baixo nível.

Especificamente, este protocolo define um conjunto de instruções-padrão para, por exemplo, executar a instalação de regras nas tabelas de roteamento, recuperar informações de tráfego, dentre muitas outras funcionalidades. Hoje em dia, corporações de grande porte da indústria de TI, como *Hewlett-Packard* (HP), CISCO *Systems*, *Nippon Electric Company* (NEC) e *International Business Machines* (IBM) produzem equipamentos de rede que operam em conformidade com o *OpenFlow*. No âmbito de controladores SDN, *frameworks* conhecidos, tais como: *Floodlight* [Big Switch Networks 2013], *Ryu* [Nippon Telegraph and Telephone Corporation 2012], POX [McCauley 2012] e *OpenDaylight* [OpenDaylight 2013] são compatíveis com o protocolo *OpenFlow*. A *Open Networking Foundation* (ONF) [ONF

2011] é o atual consórcio responsável pela sistematização e evolução deste protocolo.

No entanto, embora o *OpenFlow* tenha se consolidado como a API *southbound* padrão em arquiteturas SDN, ainda não há um consenso na forma como os controladores devem se comunicar com aplicações e usuários do plano de aplicação [Bakhshi 2017; Xia *et al.* 2015]. Hoje em dia, há um amplo conjunto de controladores; porém, cada qual define e exporta uma API *northbound* com instruções específicas para programar a rede SDN. Por exemplo, na Tabela 1.2, são mostrados exemplos de diferenças entre comandos, URIs e formatos de arquivos padrão requeridos pelas interfaces nativas de dois controladores conhecidos (*OpenDaylight* e *Floodlight*), as quais são usadas pelo gerente na operação da SDN.

Tabela 1.2: Exemplos de diferenças nos comandos das interfaces de controladores

Descrição dos exemplos de comandos	**OpenDaylight**	**Floodlight**
URI Base	/restconf/config/opendaylight-inventory:nodes/node/	/wm/staticflowpusher/json
Método HTTP padrão	PUT	POST
Formato padrão de arquivo de comandos	XML	JSON
Exemplo de comando requerido para identificar a instrução a ser instalada	<id>17</id>	"name":"17"
Exemplos de comandos requeridos para identificar o número da porta de saída de tráfego no equipamento	<instructions> <instruction> <order>0</order> <apply-actions> <action> <output-action> <output-node-connector>2 </output-node-connector> </output-action> <order>0</order> </action> </apply-actions> </instruction> </instructions>	"actions":"output=2"
Exemplos de comandos requeridos para identificar o número da porta de entrada de tráfego no equipamento	<match> <in-port>1</in-port> </match>	"in_port": "1"
Exemplos de comandos requeridos para identificar os *hosts* de origem e destino na comunicação	<match> <ethernet-match> <ethernet-source> <address>00:00:00:00:00:01</address> </ethernet-source> <ethernet-destination> <address>00:00:00:00:00:02</address> </ethernet-destination> </ethernet-match> <ipv4-source>10.0.0.1/24</ipv4-source> <ipv4-destination>10.0.0.2/24</ipv4-destination> </match>	"eth_src": "00:00:00:00:00:01" "eth_dst": "00:00:00:00:00:02" "ipv4_src": "10.0.0.1" "ipv4_dst": "10.0.0.2"
Exemplos de comandos requeridos para identificar o tipo do protocolo (e.g., TCP)	<match> <ethernet-match> <ethernet-type> <type>2048</type> </ethernet-type> </ethernet-match> <ip-match> <ip-protocol>6</ip-protocol> </ip-match> </match>	"eth_type": "2048" "ip_proto": "0x06"
Exemplo de comando requerido para identificar a prioridade da instrução a ser instalada	<priority>10</priority>	"priority":"10"

Ao observar a Tabela 1.2, primeiramente, note que as URIs para acesso aos controladores que devem ser invocadas pelo gerente para instalar configurações na rede são diferentes. Outrossim, os métodos HTTP padrão requeridos por cada controlador para aplicar instruções na rede não são semelhantes e os comandos devem ser escritos pelo gerente em um formato de arquivo específico (JSON ou XML) que depende do controlador. Além disso, ao comparar as duas últimas colunas na mesma figura, veja que muitas das estruturas de comandos requeridas pelo *OpenDaylight* ou devem ser escritas de maneira bastante diferente ou simplesmente não existem (i.e., não são suportadas) no outro controlador *Floodlight*. Verifica-se, portanto, que as interfaces *northbound* destes controladores exigem que o gerente não apenas tenha de lidar com instruções não-reutilizáveis, mas, também, tratar a variabilidade e a incompatibilidade das estruturas de programação específicas que são suportadas, para se operar ambos os controladores, o que aumenta a complexidade para se programar a rede.

Nesse contexto, os autores Rotsos *et al.* [Rotsos *et al.* 2017] afirmam que, infelizmente, não apenas a padronização de interfaces *northbound* é ainda bastante limitada, mas estas são basicamente orientadas ao *design* e, dessa forma, mais ligadas à implementação e demais características peculiares ao projeto do controlador-alvo. As diferenças entre estruturas de comandos de interfaces de controladores, mostradas na Tabela 1.2, confirmam esta condição.

Percebe-se, em particular, que a ausência de interfaces *northbound* mais independentes da plataforma de controle SDN tem levado gerentes a lidar com convenções específicas de controladores, o que aumenta a complexidade da programação da SDN e dificulta a criação de configurações de rede interoperáveis. Isso porque, como abstrações de um controlador SDN acabam normalmente não existindo em outro, múltiplas implementações de controladores resultam em múltiplas formas distintas de se programar a rede SDN [Lopes *et al.* 2016], causando dificuldades para o gerente. O resultado é que a falta de interoperabilidade das interfaces de programação para redes SDN não apenas afeta os gerentes destas redes, mas, também, as aplicações de *software* desenvolvidas para ambientes SDN, já que estas normalmente acabam por ficar acopladas a uma implementação particular de controlador.

De acordo com [Kumar e Sood 2017], um dos principais problemas relacionados à programação de redes SDN é a alta complexidade. Não é incomum que gerentes tenham, em muitos casos, de re-implementar comandos e instruções de controle já existentes, por conta da sua aplicação em um outro controlador distinto. Frequentemente, para programar

a rede e principalmente ampliar a interoperabilidade das instruções de código, gerentes precisam injetar comandos específicos de controladores SDN diretamente nos arquivos que contém as regras de controle, o que compromete a manutenibilidade das configurações de rede desenvolvidas. Diante deste cenário, os autores Rotsos *et al.* [Rotsos *et al.* 2017] citam que o desenvolvimento de interfaces *northbound* tem crescido num ritmo bastante acelerado; porém, como cada projeto desenvolve seus próprios protocolos e mecanismos de controle, a integração da interface com múltiplos controladores ainda é um desafio em aberto. Os mesmos autores destacam, ainda, que o foco na interoperabilidade de interfaces é essencial para uma orquestração de servicos de rede efetiva, flexível e com baixos custos operacionais.

Além do problema da dependência de código entre configurações da rede e controladores SDN atuais, as interfaces *northbound* destes controladores são consideradas de baixo nível [Kreutz *et al.* 2015] e provêem poucos recursos para automatizar o processo de instalação de regras na rede. Normalmente, as interfaces *northbound* disponíveis possuem grupos de comandos que, além de não serem padronizados, são difíceis de usar, pois exigem do gerente conhecimentos e habilidades não apenas de especificidades do protocolo *OpenFlow*, mas também da instalação (manual) das regras de programação da SDN, fazendo uso de comandos emitidos em interfaces de linha de comando. Por exemplo, durante todo o desenvolvimento desta pesquisa, o *framework Ryu* [Nippon Telegraph and Telephone Corporation 2012], um dos principais controladores SDN disponíveis, exige a seleção cuidadosa de cabeçalhos *OpenFlow* para formular e instalar, manualmente, regras de reserva mínima de largura de banda na comunicação entre *hosts* de uma rede SDN, conforme visto em [Ryubook 2016]—esse esforço não deveria se concentrar sobre o gerente de redes, mas desviado para a interface *northbound*, posto que os usuários do plano de aplicação superior não deveriam saber de detalhes do protocolo *OpenFlow* usado no plano de dados SDN adjacente.

Outrossim, a API [Izard 2017] do controlador *Floodlight* requer a manipulação manual e cautelosa de instruções *OpenFlow* de diferentes versões, ao invés de uma interface de alto nível, para especificar as regras que precisam ser instaladas na rede. No caso do controlador *OpenDaylight*, por exemplo, a acomodação de políticas de controle na rede SDN depende de funcionalidades (*features*) que devem ser previamente instaladas via terminal de linha de comando, como visto em [OpenDaylight 2017]. Somente após esta etapa, o gerente formulará as regras de controle da rede fazendo uso não apenas de conjuntos de convenções e instruções

incompatíveis e diferentes daqueles dos controladores há pouco citados, mas também de um formato de arquivo particular, conforme visto em [OpenDaylight 2017a]. Verifica-se, portanto, que embora os controladores *OpenFlow* sejam bastante flexíveis, não é fácil para o gerente programar uma rede SDN, considerando as implementações disponíveis.

Em 2013, um comitê técnico de trabalho [ONF 2013] foi fundado pela *Open Network Foundation* (ONF), na tentativa de padronizar possíveis categorias de interfaces *northbound*, apresentar casos de uso com protótipos envolvendo tais interfaces e tecer recomendações concretas sobre o que é necessário para potencializar o desenvolvimento de aplicações SDN para o mercado de redes de computadores nos próximos anos. Particularmente, apesar de descrições importantes quanto ao papel da interface *northbound* terem sido diretamente mencionadas pela ONF em [ONF 2014] e, mais recentemente, em [ONF 2016], ainda não há manifestações detalhadas quanto aos formatos mais apropriados para representar as informações que devem ser trocadas entre componentes SDN e as interfaces ou como modelar e implementar parâmetros de controle da rede e seus valores. Logo, apesar desta importante iniciativa da ONF, ainda não foram apresentadas interfaces *northbound* de programação de alto nível, focadas na flexibilidade e interoperabilidade. Enquanto estas soluções não emergem, infelizmente, é bastante difícil implementar aplicações e serviços que possam ser integrados em diferentes controladores SDN [Yu, Wundsam e Raju 2014].

Ao longo dos anos, estudiosos têm desenvolvido várias abordagens eficientes para implantar regras de controle e programar o comportamento da SDN, usando APIs de programação. Essas pesquisas utilizam, de um modo geral, interfaces *northbound* baseadas em padrões conhecidos, usados na especificação de APIs para serviços *Web*, como SOAP (do inglês *Simple Object Access Protocol*) [W3C 2007] e REST (do inglês *Representational State Transfer*) [Fielding 2000]. Em [Yan *et al.* 2015], por exemplo, uma abordagem voltada para o controlador *Floodlight* permite ao gerente alocar fluxos de tráfego às filas de serviço prioritário, efetuar o controle de reserva de largura de banda e computar a melhor rota de comunicação entre pontos remotos da rede SDN com mais facilidade. Mais recentemente, interfaces *northbound* têm sido propostas [Layeghy, Pakzad e Portmann 2017] priorizando a elaboração manual das instruções que serão instaladas na rede, usando um conjunto de preditores compatíveis com um único controlador [McCauley 2012].

Consequentemente, a programação do comportamento da rede SDN tem sido levada a

efeito por meio de interfaces *northbound* úteis, mas que não endereçam questões de interoperabilidade do corpo de funcionalidades existente entre controladores distintos. Verifica-se, portanto, a existência de uma lacuna na especificação de interfaces de programação para redes SDN, quando se usa abstrações não-reutilizáveis, que pode ser preenchida com interfaces mais flexíveis, que ofereçam suporte à comunicação direta com mais de um controlador.

Para lidar com essa problemática, e como contribuição principal desta pesquisa, é proposta uma interface *northbound* de programação [Vasconcelos *et al.* 2017], chamada NoBI, que prioriza o suporte à interoperabilidade, bem como acomodação simplificada e automatizada de requisitos de controle na rede, a partir de um corpo de funcionalidades uniforme e que pode ser aplicado em mais de um controlador. Em particular, a NoBI permite ao gerente reconfigurar a rede e observar seu comportamento em alto nível, usando funções agnósticas ao controlador de classe empresarial e cujas chamadas (invocações) não requerem a escrita de código específico do controlador existente. Ademais, a abordagem apresentada nesta tese não exige a formulação e instalação manual de instruções através de comandos do fabricante do equipamento de rede em terminal de linha de comando.

Em resumo, é fundamental refletir sobre a importância de se elaborar interfaces *northbound* capazes de assegurar a programação da rede SDN independentemente de controladores, em vez de impor ao gerente a tarefa de lidar com a incompatibilidade de suas interfaces. O desenvolvimento de interfaces capazes de esconder a diversidade de múltiplos controladores é, atualmente, uma necessidade [Zhang *et al.* 2018]. É nessa direção que as investigações e esforços foram conduzidos neste trabalho de doutorado.

1.2 Objetivos da Tese

1.2.1 Objetivo Principal

O objetivo principal deste trabalho é a proposição e desenvolvimento de uma interface de programação *northbound* para redes *OpenFlow* com suporte à interoperabilidade que, em particular, priorize a comunicação transparente com controladores distintos, a partir de uma arquitetura extensível e um corpo de funções uniforme.

1.2.2 Objetivos Específicos

A partir do objetivo principal deste trabalho, é possível efetuar um desdobramento para os objetivos específicos, os quais são descritos abaixo:

- definir funções agnósticas ao controlador SDN para consulta, remoção e instalação de instruções, tendo em vista programar o comportamento da rede. Além de abstrair os detalhes de operação do controlador existente, este corpo de funções deve permitir a acomodação automatizada de regras de controle, em tempo de execução;
- desenvolver componentes de *software* (*engines*) e integrá-los aos controladores SDN elencados na pesquisa, visando traduzir, corretamente, as requisições recebidas pela NoBI em instruções específicas dos controladores;
- definir um modelo de dados genérico e apropriado para o processo de representação de parâmetros de entrada para as funções NoBI. Este modelo deve ser compatibilizado com os controladores-alvo, de modo que a programação da SDN possa ser feita de forma intercambiável;
- o arcabouço do projeto da interface NoBI deve oferecer flexibilidade para ser adaptado e estendido, a fim de suportar novos controladores SDN futuros com facilidade;
- realizar uma avaliação formal confiável do funcionamento da respectiva solução proposta, utilizando cenário de rede adequado.

1.3 Justificativa e Relevância

A programação de arquiteturas de redes SDN constitui um tema bastante relevante na área de Redes de Computadores. No âmbito do mercado de redes, a realidade não é diferente. Dados de um estudo recente, realizado sistematicamente pelo *International Data Corporation* (IDC) [Kalebaila 2015], também refletem a importância das arquiteturas de redes controladas por *software*. O estudo aponta que o crescimento de mercado destas redes será de 89,4%, em 2018, com investimentos ultrapassando os 8 bilhões de dólares. Mesmo levando em conta as margens de erro associadas às previsões realizadas, é importante observar que os números apresentados pelo IDC são bastante elevados.

Muitos trabalhos encontrados na literatura, voltados para a especificação de interfaces *northbound* para redes SDN, estão focados no desenvolvimento de funções de controle com objetivos variados, mas sem apresentar os projetos das APIs propostas, quais são os parâmetros de controle disponíveis ao gerente para instalar instruções na rede ou detalhes quanto à compatibilização das funções da interface desenvolvida com o controlador escolhido.

Alguns estudos [Gorlatch, Humernbrum e Glinka 2014; Kondwilkar *et al.* 2015] baseiam-se na definição de interfaces para controlar a rede, sem especificar exatamente os projetos de tais interfaces. Outros trabalhos [Zhou *et al.* 2014] focam a interface de um *framework* controlador SDN mantido por uma única empresa, fechado e, dessa forma, pouco flexível para acomodar extensões. Em [Xu, Chen e Qian 2015], é apresentado um *framework* compatível apenas com o controlador *Ryu* [Nippon Telegraph and Telephone Corporation 2012] com uma API *northbound*, buscando o provisionamento simplificado de QoS em redes SDN. Porém, os autores não fornecem detalhes quanto aos protótipos das funções (e seus parâmetros) oferecidos ao gerente ou ainda como o corpo de funções deve ser efetivamente utilizado para programar a rede. Em [Belzarena *et al.* 2016], por exemplo, é sugerida uma arquitetura SDN de referência para controlar redes SDN, mas sem apresentar detalhes de como foi levada a efeito a avaliação de sua interface *northbound* correspondente. Há, ainda, outros estudos [Chen e Wu 2017] que propõem interfaces inspiradas em princípios de *Web* Semântica para gerenciamento de recursos da rede SDN—soluções pertinentes, mas que não admitem a pronta aplicação e uso das mesmas funções da interface em outro controlador distinto.

A literatura não é ainda conclusiva com relação à especificação de interfaces *northbound*, em termos do que é necessário ser feito para garantir a interoperabilidade de um mesmo corpo de funções entre controladores distintos e ampliar a capacidade do gerente de operar a rede de maneira mais independente da implementação do controlador existente. Assim, este tema de pesquisa apresenta-se desafiador e promissor.

Do ponto de vista das arquiteturas de redes modernas, o presente trabalho pode servir como um importante ponto de apoio na direção da automatização e redução do número de erros na instauração de políticas de controle da rede. Isso porque grande parte destas arquiteturas têm se apoiado na configuração individualizada de equipamentos distintos [Kumar e Sood 2017]. Percebe-se, em particular, a frequência com que os gerentes de redes devem elaborar e instalar um número cada vez maior de políticas de controle sofisticadas e complexas, fazendo

uso de um limitado conjunto de comandos de configuração de baixo nível (e.g., *scripts*) emitidos em uma interface de linha de comando. Nesse contexto, a utilização de uma interface *northbound* com suporte à interoperabilidade para reconfiguração automática da rede SDN é particularmente interessante para estes profissionais.

Por se tratar de um campo de pesquisa que desperta grande interesse no mercado de aplicações para redes SDN, é de extrema importância propor interfaces *northbound* capazes de controlar os recursos da rede SDN focadas na comunicação transparente com mais de um controlador. Neste contexto, o autor da tese considera que o surgimento e consolidação das "SDN *App Stores*", cujas aplicações disponíveis aos usuários não podem ficar dependentes de uma implementação particular de controlador, não será possível, se não houver esforços como os aqui despendidos, tendo em vista a especificação e avaliação de interfaces *northbound*.

Um outro aspecto que destaca a relevância da solução proposta é a sua capacidade de poder amenizar a dependência entre o controlador SDN e as configurações de rede desenvolvidas, comumente provocado pelas atuais interfaces *northbound*. Em particular, utilizando a interface proposta neste trabalho, gerentes são capazes de se comunicar com controladores distintos de forma transparente e instalar regras de controle na rede, sem que tenham de aprender a usar algum comando específico do controlador existente, lidar com detalhes da versão do protocolo *OpenFlow* ou aplicar comandos via interfaces de linha de comando.

Por fim, a comunidade científica não está indiferente às redes definidas por *software*. Em particular, um segmento crescente de pesquisadores vem defendendo não apenas a instauração das políticas reativas de gerenciamento de maneira automatizada, mas, também, o avanço das atuais interfaces para controle das redes de computadores. Dessa forma, a presente proposta de uma interface *northbound* que possibilite a programação de uma rede de forma efetiva, focalizando uma maior independência de controladores SDN e a interoperabilidade das funcionalidades da interface, tornam o presente trabalho relevante no contexto de programação de redes SDN. Até o momento da escrita deste documento, não foram encontrados outros estudos publicados na literatura com as mesmas características desta pesquisa, o que reforça o seu caráter de originalidade. Além disso, a contribuição científica deste trabalho já vem sendo respaldada pela comunidade científica, por meio da publicação de artigos científicos [Vasconcelos *et al.* 2014; Vasconcelos *et al.* 2015; Vasconcelos *et al.* 2017] em eventos de repercussão na área de redes.

1.4 Contribuições e Resultados

As principais contribuições deste trabalho são descritas a seguir:

- análise das APIs disponibilizadas por controladores SDN conhecidos, bem como a identificação dos benefícios em se conceber interfaces *northbound* mais flexíveis, tendo em vista assegurar a interoperabilidade do corpo de funções para criação de serviços e aplicações de negócio para redes SDN [Vasconcelos *et al.* 2015];

- especificação de uma interface *northbound* [Vasconcelos *et al.* 2017] alternativa, capaz de comunicar-se com controladores SDN de classe empresarial distintos e instalar requisitos de controle nos equipamentos de infraestrutura da rede. A utilização do corpo de funções da abordagem proposta não incorre em aprendizado da linguagem ou configurações específicas do controlador existente. Do ponto de vista de utilização (vide Apêndice A), o gerente apenas acessa funções (URIs) da interface NoBI proposta e envia mensagens HTTP (do inglês *Hypertext Transfer Protocol*) para orquestrar o controlador SDN existente e acomodar instruções na rede;

- construção de um modelo de dados agnóstico aos controladores SDN, para representar os parâmetros de controle da rede;

- implementação de um modelo analítico para validação de conceito em um ambiente realista de rede SDN, a fim de coletar dados experimentais e confrontar os resultados obtidos com alguns objetivos e hipóteses específicos iniciais definidos neste trabalho.

Para avaliar o potencial de uso da abordagem proposta nesta tese, um estudo experimental foi levado a efeito, cujos resultados obtidos dos ensaios apontam que as funções da interface NoBI podem efetivamente se comunicar de forma transparente com controladores distintos para programar a rede SDN. Em particular, considerando um cenário de rede SDN real, envolvendo a transmissão de vídeo entre múltiplos *hosts* e usando alguns dos melhores representantes existentes de controladores de classe empresarial [Big Switch Networks 2013; OpenDaylight 2013], observou-se que a solução proposta é capaz de compatibilizar a troca de informações entre às requisições do plano de aplicação e ambos os controladores SDN, acomodando, de forma automática, instruções de controle na rede, com um resultado operacional final satisfatório.

1.5 Organização do Documento

O restante deste documento está organizado em cinco capítulos. No Capítulo 2, são apresentados conceitos e terminologias fundamentais relacionados ao paradigma das redes definidas por *software*. Também são discutidas as principais razões que levam os gerentes a enfrentarem dificuldades para implementar serviços capazes de programar o comportamento destas redes. O Capítulo 3 apresenta ao leitor alguns trabalhos encontrados na literatura que apresentam convergência com esta pesquisa. No Capítulo 4, é apresentada uma abordagem alternativa para a programação de redes SDN, chamada NoBI, que prioriza a interoperabilidade da interface para a programação da rede e maior independência de implementações específicas de controladores. O Capítulo 5 apresenta detalhes de um estudo experimental conduzido para observar o comportamento e eficácia da solução proposta nesta tese. Por fim, o Capítulo 6 encerra este documento, onde são apresentadas as conclusões deste trabalho, um resumo das principais contribuições providas pelo mesmo, bem como direcionamentos para eventuais trabalhos futuros. Uma discussão sobre a forma de utilização de cada uma das funções oferecidas pela interface proposta, as quais foram descritas resumidamente no Capítulo 4, é levada a efeito no Apêndice A. O questionário utilizado durante a avaliação qualitativa da abordagem proposta nesta tese constitui o Apêndice B.

Capítulo 2

Redes Definidas por *Software*

Neste capítulo, são apresentados conceitos e terminologias fundamentais relacionados ao paradigma das redes definidas por *software* (SDN), os quais formam um embasamento teórico necessário para o entendimento do restante do documento.

Inicialmente, este capítulo fornece uma síntese da bibliografia de referência, situando o leitor não apenas nas práticas de gerência utilizadas em arquiteturas de redes convencionais, mas, sobretudo, discutindo as diferentes visões contidas nos paradigmas SDN e o de redes tradicionais. As dificuldades enfrentadas por gerentes de redes convencionais, os benefícios trazidos pelas ideias do modelo SDN e relatos de estudos de caso recentes, refletindo importantes experiências no emprego de princípios SDN por parte de grandes corporações de TI, também estarão sendo discutidos no decorrer deste capítulo. Em seguida, é feita uma exposição dos principais obstáculos à programação de redes SDN encarados por gerentes, considerando as interfaces *northbound* disponíveis. Serão apresentados ainda, neste capítulo, alguns dos potenciais *frameworks* controladores SDN de código aberto. Discute-se, também, princípios arquiteturais prevalentes na concepção de interfaces *northbound*. Finalmente, são apresentadas as considerações finais do capítulo.

2.1 O Modelo Tradicional das Arquiteturas de Redes

Gerenciar uma rede de computadores é uma atividade que requer *expertise* e esforço. Uma rede típica consiste não apenas de vários equipamentos distintos e de diferentes fabricantes, mas, também, de uma miríade de eventos que ocorrem simultaneamente. Nestes cenários, os

gerentes de redes são os profissionais responsáveis por formular, avaliar e implantar, eficientemente, diversas políticas para uso e controle da rede (e.g., acesso a recursos, configuração de rotas, detecção de invasões, dentre outras tarefas).

Entretanto, à medida que as demandas de usuários conectados tornam-se cada vez maiores, a habilidade destes profissionais de responder, rapidamente, a uma série de eventos gerados em tais redes tem se tornado, muitas vezes, uma tarefa demasiada complexa e até problemática. Isso acontece pois a grande maioria das arquiteturas de redes modernas apresentam poucos mecanismos de automatização para instalação de políticas de gerenciamento e programação dinâmica das operações da rede, conforme discutido em [Singh e Jha 2017]. Os mesmos autores colocam ainda que, hoje em dia, a simples inclusão de um novo dispositivo na rede é uma tarefa tediosa e suscetível a erros, por conta da diversidade de fabricantes de equipamentos e a heretogeneidade de suas interfaces de configuração, as quais exigem dos gerentes habilidades para executar procedimentos de configuração manual da rede.

Outrossim, há outros fatores que afetam de forma significativa a gerência de uma rede de computadores. Kim e Feamster [Kim e Feamster 2013], neste contexto, ressaltam que mudanças contínuas no estado atual da rede e configurações particulares aplicadas a elementos comutadores distintos são dois principais aspectos da gerência que, ainda hoje, permanecem difíceis de lidar. Além disso, muitos equipamentos de rede expõem interfaces de configuração proprietárias (fechadas) e, portanto, são operados por meio de um conjunto de comandos específicos e dependentes do fabricante em particular. Verifica-se, portanto, que a falta de mecanismos capazes de automatizar a configuração de redes e a instalação das regras de controle diretamente nos equipamentos têm prejudicado sobremaneira a evolução da infra-estrutura física, dos protocolos e até o desempenho das redes de comunicação—em especial a Internet, rotulada por alguns estudiosos na área de redes como uma rede "ossificada" [Belzarena *et al.* 2016; Xie *et al.* 2015; Feamster, Rexford e Zegura 2014].

Tradicionamente, esse efeito da "ossificação" das arquiteturas de redes convencionais pode ser atribuído principalmente à verticalização (i.e., acoplamento) entre o plano de dados—composto pelos dispositivos de redes de baixo nível—e o plano de controle da rede, cujas regras e protocolos se encontram "espalhados" [Azodolmolky 2013] nos diversos equipamentos da infraestrutura física da rede. É claro que, no que diz respeito à configuração de redes, a escolha das práticas de controle da rede por parte do gerente pode variar de organização para

organização. Além disso, as redes de computadores convencionais podem apresentar uma variedade de características, com maiores ou menores diferenças. Entretanto, é importante frisar que, atualmente, elas possuem um ponto em comum: as instruções de controle da rede são fortemente acopladas aos equipamentos de sua infraestrutura [Hu, Hao e Bao 2014].

Ainda nos dias atuais, é bastante comum encontrar gerentes de redes fazendo uso de técnicas *ad hoc* (e.g., formulação de *scripts*) para configuração manual e individualizada de dispositivos da infraestrutura da rede. Entretanto, deve-se observar que esta prática é inadequada, uma vez que tais *scripts* possuem grupos de comandos que, além de não serem padronizados, são difíceis de serem reutilizados em outros cenários de redes que possuam características distintas. Por isso, elaborar e sobretudo instalar, apropriadamente, diversas políticas efetivas de controle da rede fazendo uso de um conjunto de comandos de baixo nível aplicados em diversos equipamentos de fabricantes distintos têm se tornado tarefas não-triviais para gerentes de redes convencionais, e devem, então, ser facilitadas.

Diante da complexidade e rigidez das arquiteturas de redes tradicionais [Ghodsi *et al.* 2011], pesquisas recentes têm formulado técnicas, visando não somente implantar políticas de controle da rede de maneira automatizada, mas, sobretudo, expressá-las usando linguagens de alto nível [Trois *et al.* 2016; Foster *et al.* 2013] e outras abstrações [Casado, Foster e Guha 2014; Reitblatt *et al.* 2012]. Em particular, as "redes programáveis" têm sido apontadas por estudiosos como um caminho bastante interessante para potencializar a inovação e evolução das redes, além de simplificar a instalação e manutenção das políticas de controle.

2.2 O Modelo de Redes Definidas por *Software*

Conforme foi exposto no capítulo anterior, há um grande interesse por parte de pesquisadores de todo o mundo em observar as arquiteturas de rede definidas por *software* (SDN, do inglês *Software-Defined Networks*). Em linhas gerais, este recente paradigma propõe uma abordagem inovadora, a fim de controlar e reconfigurar o comportamento global de uma rede de computadores: retirar a lógica de controle da infraestrutura da rede, centralizá-la em um sistema operacional de rede (ou *framework* controlador) e transformar os equipamentos de rede da SDN em meros comutadores de pacotes, os quais serão orquestrados por meio de uma interface não-proprietária. O "cérebro" de uma rede SDN estará, portanto, completamente

desacoplado dos vários dispositivos de baixo nível da rede—o que representa uma ruptura expressiva da tradicional dependência entre os planos de controle e dados observada nas arquiteturas de redes convencionais.

O paradigma SDN teve seu início em experimentos acadêmicos levados a efeito no *campus* da Universidade de *Stanford* [McKeown *et al.* 2008; Greene 2009]. Apesar de mostrar-se um paradigma recente e apresentar uma proposta bastante inovadora, as ideias e tecnologias precursoras deste modelo não foram estabelecidas nesta década. Na verdade, as arquiteturas de redes definidas por *software* têm suas raízes em pesquisas anteriores, que envolviam a programabilidade do plano de dados da rede, tais como Redes Ativas (do inglês *Active Networks*) [Tennenhouse *et al.* 1997; Calvert *et al.* 1998]; a separação entre os planos de controle e dados da rede, como ForCES [Yang *et al.* 2004] e RCP (do inglês *Routing Control Plataform*) [Caesar *et al.* 2005]; a virtualização de redes, como MBone [Macedonia e Brutzman 1994] e *Tempest* [Merwe *et al.* 1998]; e a concepção dos primeiros *frameworks* de gerência, a exemplo de Cisco IOS [Bollapragada, Murphy e White 2000] e NOX [Gude *et al.* 2008]. Mais recentemente, outros esforços têm sido empreendidos na separação completa entre o plano de controle e de dados da rede, como é o caso de *OpenFlow* [McKeown *et al.* 2008] e POF [Song 2013]; enquanto que POX [McCauley 2012], ONOS [Berde *et al.* 2014], *Floodlight* [Big Switch Networks 2013], *OpenDayLight* [OpenDaylight 2013] e *Ryu* [Nippon Telegraph and Telephone Corporation 2012] são projetos que têm se dedicado ao estudo e desenvolvimento de sistemas operacionais de rede.

Nos últimos anos, além da academia, o modelo de redes definidas por *software* tem despertado a atenção de importantes corporações de TI e comunidades ligadas à indústria de redes. Recentemente, empresas conhecidas, como *Facebook*, *Yahoo! Inc.* e *Microsoft Corporation*, aliadas a outros grupos de fabricantes de equipamentos e provedores de serviços para redes, fundaram a *Open Networking Foundation* (ONF) [ONF 2011], com o intuito de promover as redes programáveis e sistematizar a evolução do protocolo *OpenFlow*.

Em 2013, a empresa americana *Google*, por exemplo, apresentou um estudo [Jain *et al.* 2013] pioneiro que nos dá uma ideia clara acerca da incorporação de princípios do modelo SDN no projeto, implantação e avaliação de uma arquitetura SDN global para interconectar centros de dados distribuídos ao redor do mundo: os autores afirmam que, em apenas três anos, foi possível não somente observar taxas de eficiência de comunicação até 3 (três) vezes superiores àquelas obtidas anteriormente, mas, também, reduzir custos operacionais.

Um outro estudo de caso, desenvolvido pela *Microsoft Corporation* [Russinovich 2015], também reforça os benefícios oriundos do paradigma SDN. A empresa cita que, para manter a escalabilidade de sua intraestrutura global de provisionamento de serviços de computação na nuvem, com elevadíssimas demandas de computação e armazenamento que duplicam a cada 6 (seis) meses, recorreu às técnicas de centralização de controle e separação entre os planos de controle e dados, para que diversos componentes da infraestrutura de rede em si pudessem ser modificados e programados, mais rapidamente.

Verifica-se, portanto, que um número cada vez maior de empresas está começando a considerar alguns dos princípios e valores fundamentais do paradigma SDN como uma estratégia efetiva de negócio.

O interesse pela utilização de controladores SDN tem crescido muito nos últimos anos, por reduzir sobremaneira as complexidades associadas à formulação e instalação das regras de gerência nos dispositivos que formam a infraestrutura de rede, conforme discutido no capítulo anterior. Em particular, estes *frameworks* fornecem abstrações que permitem ao gerente de redes recuperar informações importantes sobre o estado atual da rede SDN, que incluem, por exemplo, estatísticas de tráfego e congestionamento. De certa forma, os controladores desempenham um papel similar àquele dos sistemas operacionais tradicionais: são capazes de abstrair os detalhes de funcionamento dos diferentes dispositivos de baixo nível da rede.

Da perspectiva do administrador de redes SDN, um *software* controlador é o responsável, por exemplo, por coletar diferentes informações sobre o estado global da rede e reportá-las de maneira simplificada, abstraindo toda a complexidade de sua infraestrutura. Além disso, as abstrações fornecidas por este *framework* devem permitir modificar o comportamento da rede automaticamente; evitando, portanto, que se tenha de manualmente instalar (ou desinstalar) regras de controle em cada um dos equipamentos da rede envolvidos.

Ultimamente, as ideias do paradigma SDN estão deixando de ser uma eventualidade no contexto de projeto de redes para se tornar uma realidade, agregando valor competitivo à indústria de redes e trazendo alguns benefícios, tais como: (1) redução da complexidade na implantação de políticas de controle da rede; (2) recuperação simplificada de informações de baixo nível sobre o estado atual da rede; (3) configuração do comportamento da rede por meio de abstrações de mais alto nível fornecidas por *frameworks* controladores; e (4) redução do número de erros na configuração manual de dispositivos da infraestrutura da rede.

Por fim, Sahoo *et al.*, em [Sahoo *et al.* 2017], descrevem o modelo SDN como sendo o futuro das redes de computadores. O fato de vários trabalhos de pesquisa recentes e algumas das gigantes da tecnologia fazerem alusão às ideias e princípios das arquiteturas de redes definidas por *software* é uma clara indicação de mudanças no paradigma tradicional de redes em todo o mundo. Contudo, deve-se observar que, nesta busca pela simplificação da instalação de regras de controle na SDN, o desenvolvimento de interfaces de programação *northbound* agnósticas ao controlador existente e com suporte à programação em alto nível da SDN tem sido deixado em segundo plano, fazendo, portanto, com que gerentes encontrem inúmeras dificuldades para implementar serviços para estas redes.

2.3 O Desafio de Programar Redes Definidas por *Software*

Em arquiteturas de redes definidas pof *software*, a comunicação entre as aplicações de *software* e o controlador é levada a efeito por meio de uma interface de programação denominada API *northbound*, enquanto que uma API *southbound* permite a interação entre o *framework* SDN e os equipamentos da infraestrutura da rede [Haleplidis *et al.* 2015]. O plano de aplicações, a priori, depende da interface *northbound* do controlador, para programar a rede.

Apesar da interface *southbound* se encontrar bem consolidada com o *OpenFlow* [McKeown *et al.* 2008], ainda não há um consenso na forma como diferentes controladores devem se comunicar com entidades do plano de aplicações SDN na camada mais alta [Daradkeh *et al.* 2016; Nunes *et al.* 2014]. Hoje, existem vários controladores SDN, cada qual com uma interface *northbound* de controle específica, mas nenhum deles detém *market share* significativo. Em particular, percebe-se, nos projetos que desenvolvem estes *frameworks* controladores, uma certa autonomia no tocante à especificação de suas interfaces *northbound*, o que complica sobremaneira a interoperabilidade e garantia de comunicação transparente da interface desenvolvida com outros diferentes controladores. Por exemplo, há interfaces que exigem do gerente a escrita de comandos específicos, como aquelas encontradas nos controladores NOX [Gude *et al.* 2008] e Maestro [Cai, Cox e Ng 2011], interfaces *northbound* baseadas em padrões usados na especificação de APIs para serviços *Web*, como SOAP (do inglês *Simple Object Access Protocol*) [W3C 2007], e outras baseadas no padrão REST (do inglês *Representational State Transfer*) [Fielding 2000; Richardson e Ruby 2008].

Além do problema da ausência da padronização de categorias de interfaces *northbound*, à medida que o número de controladores SDN tem aumentado, a habilidade do gerente de poder utilizar estruturas de comandos de uma interface em diferentes controladores é bastante reduzida. Por exemplo, a Figura 2.1 mostra as diferenças nas estruturas de comandos usados para a elaboração de uma mesma regra para programação da SDN visando o encaminhamento do tráfego IP gerado na porta de origem (`1`) e destinado à porta (`2`) do equipamento *OpenFlow*, na comunicação entre dois *hosts* da rede com endereços de origem (`192.168.1.1`) e destino (`192.168.1.2`) em dois controladores SDN distintos:

OpenDaylight

```
URI: /restconf/config/opendaylight-inventory:nodes/node/<switch-id>/
table/0/flow/17
HTTP PUT

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
  <instructions>
      <instruction>
         <order>0</order>
         <apply-actions>
             <action>
                 <output-action>
                    <output-node-connector>2</output-node-connector>
                 </output-action>
                 <order>0</order>
             </action>
         </apply-actions>
      </instruction>
  </instructions>
  <table_id>0</table_id>
  <id>17</id>
  <match>
      <ethernet-match>
          <ethernet-type>
              <type>0x800</type>
          </ethernet-type>
          <ethernet-source>
              <address>00:00:00:00:00:01</address>
          </ethernet-source>
          <ethernet-destination>
              <address>00:00:00:00:00:02</address>
          </ethernet-destination>
      </ethernet-match>
      <ipv4-source>192.168.1.1/24</ipv4-source>
      <ipv4-destination>192.168.1.2/24</ipv4-destination>
      <in-port>1</in-port>
  </match>
  <hard-timeout>100</hard-timeout>
  <idle-timeout>75</idle-timeout>
  <flow-name>FROM-IP-1-TO-IP-2</flow-name>
  <priority>11</priority>
</flow>
```

Floodlight

```
URI: /wm/staticflowpusher/json
HTTP POST

{
  "switch":"00:01:cc:3e:5f:81:ea:01",
  "name":"FROM-IP-1-TO-IP-2",
  "priority":"11",
  "idle_timeout": "75",
  "hard_timeout": "100",
  "active":"true",
  "in_port": "1",
  "eth_src": "00:00:00:00:00:01",
  "eth_dst": "00:00:00:00:00:02",
  "eth_type": "0x0800",
  "ipv4_src": "192.168.1.1/24",
  "ipv4_dst": "192.168.1.2/24",
  "actions":"output=2"
}
```

Figura 2.1: Exemplo de composição de uma mesma regra para programação da SDN com diferentes estruturas de comandos requeridas pelas APIs de controladores

Observe, na Figura 2.1, que os controladores *OpenDaylight* [OpenDaylight 2013] e *Floodlight* [Big Switch Networks 2013] definem suas próprias funções (URIs REST) de controle, formatos de arquivo padrão (JSON ou XML), modelos de representação de dados (e.g., nomenclatura de *tags* e campos) e instruções. Ainda, no *OpenDaylight*, uma URI e uma requisição HTTP com o método PUT serão usados para instalar a regra elaborada na SDN,

enquanto que o *Floodlight* espera o HTTP POST e uma URI distinta para a mesma finalidade.

Verifica-se, pois, que a diversidade de controladores e dialetos de suas interfaces pode se tornar a principal barreira para a reutilização de funcionalidades e da programação da rede SDN em múltiplos controladores. Posto que estes *frameworks* utilizam linguagens, configurações e modelos de representação de dados heterogêneos, há, em muitos casos, que se reescrever partes inteiras de uma instrução de controle da rede já existente, por conta apenas da sua aplicação (instalação) em um outro controlador SDN distinto. Salman *et al.* [Salman *et al.* 2016], nesse contexto, afirmam que uma das razões para esta condição é que as APIs dos controladores modernos estão longe de ser independentes das estruturas da linguagem de programação utilizada em sua implementação. Os mesmos autores destacam, ainda, que para formular e instalar instruções de rede em diferentes controladores, estes requerem que o gerente esteja ciente da variabilidade entre cada uma das estruturas de comandos suportadas por cada controlador, o que certamente demanda tempo e nem sempre é fácil de alcançar.

Diante deste cenário, um comitê técnico de trabalho [ONF 2013] foi fundado pela *Open Networking Foundation* (ONF), com vistas à concepção e, principalmente, padronização de interfaces *northbound* para redes SDN (seguindo, de certa forma, um caminho similar àquele utilizado para promover a padronização da API *southbound* com *OpenFlow*). Este comitê busca explorar as implementações necessárias para garantir a interoperabilidade entre controladores e aplicações de *software* [Schneider *et al.* 2014]. Há, ainda, trabalhos encontrados na literatura (descritos no Capítulo 3) voltados para a especificação de interfaces *northbound* para redes SDN. Entretanto, até o momento da escrita deste documento, não foram apresentadas implementações de interfaces *northbound* focadas nas questões de garantia de interoperabilidade das funcionalidades disponíveis ao gerente e comunicação transparente da interface com controladores distintos.

A respeito desta iniciativa da ONF, alguns estudiosos apontam que o comitê técnico fundado pode enfrentar um grande desafio para consolidar uma única interface *northbound* padrão capaz de assegurar a interoperabilidade de uma dada classe de serviços entre os principais controladores SDN do mercado [Lopes *et al.* 2016; Metzler 2014; Pitt 2013]. Nesta mesma linha de raciocínio, há outros trabalhos [Daradkeh *et al.* 2016; Kreutz *et al.* 2015] que afirmam que diferentes implementações de interfaces irão emergir; porém, a indústria de redes SDN é quem selecionará quais conjuntos de APIs tornar-se-ão padrões *de facto*.

Tijare [Tijare 2016], ao abordar as diferentes perspectivas entre a ONF e membros da comunidade redes, destaca que uma razão para a ausência de interfaces *northbound* consolidadas se deve à própria cultura da normatização de protocolos. Em particular, as condutas de padronização levadas a efeito ao longo dos anos por comitês técnicos regulatórios, como IETF[1] (do inglês *The Internet Engineering Task Force*), são diferentes daquelas normalmente empregadas pelas comunidades. No primeiro caso, por exemplo, os investimentos e a especificação de padrões e normas são levados a efeito *antes* da etapa de implementação; ao passo que, em comunidades de usuários, as implementações de APIs, *kits* de desenvolvimento de sistemas e demais soluções de *software* emergem primeiro. A partir daí, a competição aberta de mercado irá determinar quais implementações virão a prosperar e se estabelecer.

Com base em toda esta exposição, pode-se perceber que, apesar da ONF e membros da comunidade de redes abordarem a mesma atividade—a padronização de interfaces *northbound*—, eles operam com perspectivas bastante diferentes. Alguns [Hawley 2013] defendem que um grande número de interfaces *northbound* poderá acarretar uma "*falta de consistência*" que afetará negativamente os desenvolvedores de aplicações, enquanto que outros estudiosos [Daradkeh *et al.* 2016] afirmam que a competição de mercado é bem vinda e que apenas um estreito conjunto de interfaces *northbound*, voltadas para um escopo reduzido de controladores, "*irá sobreviver*". De certa forma, tal dicotomia entre estas visões torna-se, infelizmente, uma barreira para o desenvolvimento e consolidação de interfaces *northbound*.

Um outro obstáculo à programação de redes SDN reside no fato de que várias interfaces *northbound* encontradas na literatura propõem parâmetros de controle da rede bastante variados, porém, dependentes de instruções de protocolos disponíveis nos equipamentos da rede, conforme discutido em [Sieber *et al.* 2015; Caba e Soler 2015]. Isso significa que funcionalidades da interface *northbound* podem, em muitos casos, não ser totalmente suportadas pelo *firmware* padrão do equipamento, sobretudo quando uma versão mais recente do controlador é liberada. O resultado é que gerentes devem primeiro verificar se determinado recurso da interface é (ou não) implementado pelo protocolo *southbound* da rede e ainda suportado pelo equipamento em si—isto não apenas tem um impacto direto na utilização da interface, mas, também, cria dificuldades para o gerente formular e acomodar regras de controle que possam ser executadas em diferentes controladores.

[1] <https://www.ietf.org>

O leque de recursos de controle oferecido pela interfaces *northbound* é crítico para a aceitação de um controlador de *software* por parte do gerente de redes. Se um controlador SDN não provê suporte adequado à programação de redes, provavelmente será rejeitado. É possível que haja, também, uma má utilização das funções de controle oferecidas pela interface e o descontentamento dos seus utilizadores. Vale ressaltar que, mesmo que uma interface *northbound* de um controlador esteja sendo utilizada pelo gerente, isso não significa necessariamente que ela seja flexível e suas funções possam ser imediatamente reutilizadas em diferentes cenários de redes SDN contendo outro controlador. E importante destacar que existem outros aspectos que condicionam o uso de uma interface *northbound* de um controlador, a exemplo da capacidade de escolha do(s) seu(s) usuários(s), baseada em opções de soluções similares e até mesmo dos custos associados às escolhas.

Com isso, torna-se necessária a busca por outras interfaces *northbound*, mais focadas na independência de controlador. Os autores Blial, Mamoun e Benaini [Blial, Mamoun e Benaini 2016], nesse contexto, destacam que enquanto estas soluções de mais alto nível não emergem, gerentes de redes acabam por não conseguir integrar serviços e aplicações à arquiteturas SDN com interfaces incapazes de suportar múltiplos controladores.

Muitos são os benefícios ganhos em desenvolver interfaces de programação *northbound* que assegurem a portabilidade de configurações de rede e de código. Em particular, tais soluções permitiriam, por exemplo, que se pudesse programar a rede SDN de maneira simplificada, a partir do reuso de instruções de controle previamente elaboradas, em vez de re-implementá-las por completo, além de aumentar a produtividade e flexibilidade operacional [Doriguzzi-Corin *et al.* 2016]. Adicionalmente, conforme se discute em [Vasconcelos *et al.* 2015], o estudo de tais interfaces alternativas certamente criaria novas oportunidades para boa parte da comunidade de redes, a exemplo do surgimento e consolidação de lojas de aplicativos voltadas especificamente para redes SDN—i.e., as *"SDN App Stores"*.

Recentemente, a *Hewlett-Packard* (HP) anunciou o lançamento de sua *SDN App Store* [Hewlett-Packard 2013a; Duckett 2013]. A empresa cita que os clientes HP poderão buscar, adquirir e, em alguns casos, comprar aplicações de *software* para aquelas redes SDN que utilizem o controlador que é mantido por ela: o HP *Virtual Application Networks SDN Controller* [Hewlett-Packard 2013b; Hewlett-Packard 2016]. Embora reconhecendo o pioneirismo da HP na concepção de uma loja de aplicativos especificamente voltados para redes SDN, o

modelo de negócio da HP constitui um obstáculo aos gerentes e projetistas de *software* que queiram se orientar através deste modelo para criar e disponibilizar serviços e aplicações para redes SDN. Isso porque, em linhas gerais, a estratégia da HP pressupõe que estes profissionais utilizem um *kit* de desenvolvimento de *software* (SDK) [Hewlett-Packard 2016b] específico, juntamente com um controlador SDN proprietário, cujos custos associados às modalidades de licença de uso podem variar de $495 até $9.990 dólares [Hewlett-Packard 2014]. O *Gartner Group* [Gartner Inc. 2014], nesse contexto, destaca que, a fim de modificar o futuro das redes de computadores e alcançar o verdadeiro potencial do mercado de aplicações para redes SDN, a HP deve compatibilizar sua plataforma SDN com um controlador aberto [OpenDaylight 2013], tornando-a, assim, disponível a um número maior de envolvidos. Dentre outras limitações desta plataforma SDN da HP, o *Gartner Group* coloca ainda que as aplicações que compõem a loja virtual só podem ser executadas juntamente com o controlador da HP, o que compromete seriamente a migração destes aplicativos para outros ambientes de redes SDN com características e controlador distintos.

Com base no contexto apresentado nesta seção, fica claro que é essencial desenvolver interfaces *northbound* uniformes e flexíveis, que possibilitem programar uma rede SDN de maneira simplificada e automática, sem resultar em acoplamento indesejado de código e capazes de suportar múltiplos controladores. A ausência de interfaces de programação com tais características aumenta a demanda por pesquisas neste assunto. Ainda, o foco em controladores SDN de classe empresarial gratuitos e de código aberto certamente possibilitará, a um maior número de interessados que não dispõem de recursos antecipados e significativos, melhores condições para criar serviços com valor de negócio para redes SDN e ingressar no mercado de aplicações voltadas para estas redes, mais rapidamente.

2.3.1 *Frameworks* Controladores para Arquiteturas SDN

Nesta seção, serão apresentados alguns dos principais *frameworks* controladores de gerência de redes de código aberto e de classe empresarial na área de Redes Definidas por *Software*. Serão discutidos o funcionamento das interfaces *northbound* de controle oferecidas por estes controladores, suas características e possíveis limitações no processo de instauração de regras de controle da rede. Além disso, delimita-se o escopo do suporte ferramental usado neste trabalho em relação a cada um destes *frameworks*.

Em uma rede definida por *software*, o controlador tem o papel de instalar, automaticamente, as regras de controle nos equipamentos que formam o plano de dados da rede, orquestrando-os. Em particular, tais equipamentos recebem as instruções oriundas do controlador por meio de uma API *southbound* não proprietária, como *OpenFlow* [McKeown *et al.* 2008], por exemplo. Além de comandar os dispositivos da infraestrutura da rede, o controlador também é capaz de coletar e repassar, para o plano de aplicações, informações importantes sobre o estado da rede—suprimindo detalhes internos de funcionamento dos equipamentos e abstraindo, assim, a complexidade de sua infraestrutura física. Por outro lado, as aplicações de *software* e os serviços de rede de mais alto nível comunicam-se com o controlador por meio de uma outra interface, denominada API *northbound*. O plano de aplicação deve utilizar, portanto, a interface *northbound* que é fornecida pelo controlador SDN, a fim de programar a rede.

Ao longo dos anos, a constante necessidade de maior flexibilidade, redução dos custos relacionados à aquisição de ferramentas e adaptabilidade em arquiteturas de redes SDN tem alavancado o surgimento de diversos *frameworks* controladores gratuitos e de código aberto. Dados de uma pesquisa inicial, realizada pelos autores Khondoker *et al.* [Khondoker *et al.* 2014], mostram que estes *frameworks* de gerência para arquiteturas SDN emergiram com alta flexibilidade e já começam a ser utilizados largamente. Em particular, na busca pelo melhor controlador gratuito e de código aberto existente, cinco foram inicialmente selecionados na pesquisa citada: POX [McCauley 2012], *Ryu* [Nippon Telegraph and Telephone Corporation 2012], *Trema* [Takamiya e Karanatsios 2012], *FloodLight* [Big Switch Networks 2013] e *OpenDaylight* [OpenDaylight 2013]. Para cada controlador, dez propriedades foram analisadas: (a) qualidade geral de suas interfaces, (b) suporte geral, (c) interface gráfica (GUI) do utilizador, (d) REST API, (e) produtividade, (f) documentação, (g) modularidade, (h) tempo de projeto, (i) compatibilidade com o *OpenFlow* e (j) suporte ao *Neutron* [Neutron 2012] a fim de atuar como possível *backend* (estrutura de retaguarda). Para cada propriedade, foi atribuído um peso, indicando, portanto, o impacto que cada uma possui na escolha do controlador. Neste sentido, o estudo estabelece uma espécie de "ranking", o qual pode ser observado por organizações e seus gerentes de redes a fim de promover um processo de tomada de decisão mais informado. Os resultados da pesquisa [Khondoker *et al.* 2014] apontam que, dentre os *frameworks* estudados, o *Ryu* se destaca como sendo o melhor controlador gratuito de código aberto; seguido pelo *FloodLight* no segundo posto.

Os autores Khattak, Awais e Iqbal [Khattak, Awais e Iqbal 2014] propuseram um dos primeiros estudos comparativos envolvendo os controladores *OpenDaylight* [OpenDaylight 2013] e *Floodlight* [Big Switch Networks 2013], considerando o tempo (médio) gasto para processar pacotes e o número de pacotes processados em um dado intervalo de tempo como principais métricas de desempenho. Nesta pesquisa, os autores afirmam que o *FloodLight* apresentou desempenho bastante consistente e superior ao outro controlador em termos do tempo gasto para processar pacotes na rede. Outrossim, em testes de carga conduzidos para observar a quantidade de pacotes processados por segundo, o *OpenDaylight* apresentou resultados inferiores ao *FloodLight*. Segundo os autores, uma razão provável para este desempenho desfavorável do *OpenDaylight* se deve a problemas no gerenciamento de memória e ainda mal funcionamento deste controlador influenciado por *plugins* inconsistentes.

Mais recentemente, os autores Salman *et al.* [Salman *et al.* 2016] apresentam um estudo interessante e proporcionam uma comparação bastante completa e coerente de *frameworks* SDN (em termos do número de controladores comparados, quantidade de critérios observados, discussão de resultados, etc.). Neste estudo, além de investigar um amplo número de candidatos—(12) doze—envolvendo controladores recentes e mais antigos, a pesquisa utiliza um número de critérios de comparação superior a outros trabalhos [Khondoker *et al.* 2014] anteriores. Resumidamente, os autores destacam os controladores MUL [MUL 2014], *Maestro* [Cai, Cox e Ng 2011] e *Beacon* [Erickson 2013] como os que apresentam melhor desempenho em termos do tempo médio gasto para processar pacotes, ao passo que *Ryu* [Nippon Telegraph and Telephone Corporation 2012] e POX [McCauley 2012] apresentaram os piores índices neste critério. Ainda, afirmam que os controladores ONOS [Berde *et al.* 2014] e *OpenDaylight* [OpenDaylight 2013] são os que possuem maior número de recursos (*features*), enquanto que o *Ryu* [Nippon Telegraph and Telephone Corporation 2012] não detém boa modularidade. Além de posicionar o *OpenDaylight* como sendo o controlador do futuro, os autores citam que sua documentação é superior àquela oferecida pelo ONOS. Por fim, a pesquisa aponta que a escolha do controlador deve ser baseada levando-se em conta diversos critérios, além dos requisitos dos usuários.

Em [Valluvan, Manoranjitham e Nagarajan 2016], é apresentada uma proposta para avaliação de múltiplos controladores: NOX [Gude *et al.* 2008], POX [McCauley 2012], *Beacon* [Erickson 2013], *Floodlight* [Big Switch Networks 2013], MUL [MUL 2014], *Open-*

Daylight [OpenDaylight 2013], *Maestro* [Cai, Cox e Ng 2011] e *Ryu* [Nippon Telegraph and Telephone Corporation 2012]. Neste estudo, os autores posicionam o *Floodlight* junto aos demais como o controlador SDN que apresenta melhor suporte à operação de redes SDN de grande porte, enquanto que o *OpenDaylight* é apontando como sendo o controlador do futuro.

Uma análise particular do *OpenDaylight* [OpenDaylight 2013] foi levada a efeito, em [Suh *et al.* 2016], tendo em vista verificar o comportamento e desempenho deste *framework* em ambiente SDN de múltiplas instâncias em *cluster*. Neste trabalho, os pesquisadores destacam que este controlador apresenta uma boa capacidade de recuperação à falhas.

Diferentemente de estudos que analisam o desempenho de controladores usando métricas tradicionais quantitativas (e.g., tempo médio para o processamento de pacotes e quantidade de pacotes processados), a pesquisa desenvolvida em [Bondkovskii *et al.* 2016] investiga a facilidade de uso percebida e a maturidade das interfaces *northbound* do *OpenDaylight* [OpenDaylight 2013] e ONOS [Berde *et al.* 2014] em um caso de uso específico. Para cada controlador, quatro critérios qualitativos foram observados: a) facilidade para descobrir a topologia de rede; b) facilidade para instalar instruções; c) facilidade para alterar, em tempo de execução, uma configuração ativa; e, por fim, facilidade para remover uma configuração do controlador. Os resultados da pesquisa são desfavoráveis, para ambos os controladores. Em particular, os autores reportam que a API do *OpenDaylight* não é bem documentada e o processo de instalação e remoção de instruções é árduo; principalmente se o utilizador for inexperiente. Os autores citam, ainda, que o mesmo se aplica a interface REST do ONOS. Com base nestas conclusões, é possível inferir que, apesar de algum progresso ter sido levado a efeito, técnicas de usabilidade podem não ter sido adequadamente empregadas nos projetos destes controladores, com vistas à melhoria da qualidade geral de suas interfaces de uso.

Os autores Yamei, Qing e Qi, em [Yamei, Qing e Qi 2016], conduziram uma análise comparativa entre os controladores SDN *OpenDaylight* [OpenDaylight 2013] e ONOS [Berde *et al.* 2014]. Neste trabalho, em particular, as seguintes métricas de comparação foram usadas: a) tempo médio para o processamento de pacotes, b) quantidade de pacotes processados e c) capacidade de estabelecimento de múltiplos canais de comunicação (*channel capacity*). Neste estudo, os autores afirmam que o *OpenDaylight* possui desempenho geral superior ao ONOS. O primeiro controlador destaca-se por ser mais flexível, apresenta melhor estabilidade e o processo de instalação e remoção de novos serviços (via *plugins*) é mais simples do que

no ONOS. Além disso, o tempo médio para se estabelecer uma conexão estável com os dispositivos de rede no *OpenDaylight* é inferior àquele observado com o ONOS. O estudo reforça, ainda, que ONOS possui uma GUI intuitiva, enquanto que o *OpenDaylight* se destaca no processo de mapeamento e descoberta da topologia SDN.

Outra pesquisa recente [Arbettu *et al.* 2016] propõe um estudo comparativo entre os *frameworks OpenDaylight* [OpenDaylight 2013], ONOS [Berde *et al.* 2014], *Rosemary* [Shin *et al.* 2014] e *Ryu* [Nippon Telegraph and Telephone Corporation 2012], tomando como foco principal a segurança. Em particular, os autores deflagraram diversos ataques de segurança conhecidos a cada um destes controladores, e análise dos resultados finais obtidos aponta que o *OpenDaylight* se distancia dos demais sendo, portanto, mais seguro e tolerante à invasões.

Em [Karakus e Durresi 2017], são apresentados estudos que tem por base o levantamento do estado da arte das pesquisas realizadas na área de QoS com redes SDN nos últimos anos. Em seus estudos, os autores citam que a maioria destas pesquisas escolheu o *Floodlight* como controlador primário não apenas por oferecer melhor suporte às operações de QoS sobre outros controladores, mas, também, por sua boa modularidade e um rico conjunto de APIs.

Os resultados destes estudos anteriores constituíram um elemento-chave para a escolha de potenciais controladores a serem utilizados em arquiteturas de redes SDN. Nesse sentido, foram selecionados, para realização deste trabalho, os *frameworks FloodLight* e *OpenDaylight*. Estes controladores SDN serão descritos a seguir.

Floodlight

Hoje em dia, um dos controladores SDN mais utilizados é o *Floodlight* [Big Switch Networks 2013]. Ele teve seu desenvolvimento iniciado em 2012, a partir de uma ramificação (*"fork"*) do código-base de outro controlador—*Beacon* [Erickson 2013]. Já em sua primeira versão, o *Floodlight* apresentou diversas funcionalidades importantes e corrigiu algumas limitações associadas ao *Beacon*, tornando-o, assim, bastante popular.

Além de ser considerado um controlador SDN de classe empresarial, o código-fonte do *Floodlight* é aberto e distribuído sob a licença *Apache*. O *Floodlight* possui suporte ao protocolo *OpenFlow*, pode ser utilizado como *backend* (estrutura de retaguarda) para plataformas de computação na nuvem (e.g., *OpenStack* [OpenStack 2014]) e é um dos controladores de código aberto mais bem documentados. De acordo com o *site* do projeto [Big

Switch Networks 2013], o *Floodlight* é atualmente mantido por uma ampla comunidade ativa de desenvolvedores autônomos, juntamente com engenheiros da empresa *Big Switch Networks* [Big Switch Networks 2016], que financia o projeto.

Um diferencial entre o *Floodlight* e muitos outros controladores SDN existentes é que, por ser um *framework* baseado na linguagem de programação *Java*, o *Floodlight* pode ser facilmente executado em diferentes sistemas operacionais (e.g., Linux, Mac OS e *Windows*).

Entretanto, apesar dos esforços do projeto *Floodlight* com vistas à definição de uma interface *northbound* estruturada e orientada a princípios arquiteturais conhecidos (vide Subseção 2.3.2), pode-se dizer que sua API é considerada como sendo de baixo nível e oferece poucos recursos de automatização para programar a SDN. Por exemplo, analisando a API [Izard 2017] deste do controlador, percebe-se o gerente frequentemente deverá efetuar uma manipulação (manual) e cautelosa de instruções de diferentes versões *OpenFlow*, para especificar as regras de controle que precisam ser instaladas na rede.

OpenDaylight

O *OpenDaylight* é um controlador SDN gratuito e suportado não apenas por grandes corporações, como IBM, *Cisco* e *Juniper*, mas, também, por uma ampla comunidade de usuários. A primeira versão deste controlador de classe empresarial, chamada *Hydrogen*, foi lançada no início de 2014 e, atualmente, o *OpenDaylight* é parte integrante da *Linux Foundation*.

Conforme mencionado no início da seção, o *OpenDaylight* é apontado por estudiosos [Valluvan, Manoranjitham e Nagarajan 2016; Salman *et al.* 2016] como sendo o controlador SDN do futuro, além de ser um dos melhores de código aberto existentes, superando muitos outros concorrentes. O controlador é inteiramente baseado na linguagem de programação *Java* e provê amplo suporte às diferentes versões do protocolo *OpenFlow*.

Da mesma forma como acontece no *Floodlight*, o *OpenDaylight* define e mantém sua própria interface *northbound* de programação da rede. Em particular as principais funcionalidades do controlador estão consolidadas em uma interface *Web*, chamada DLUX [DLUX 2017]. Pode-se utilizar URIs no formato REST [Fielding 2000] para instalar instruções na SDN [OpenDaylight 2017a]. Além disso, o gerente também dispõe de uma GUI particular, chamada YANG, que permite programar e recuperar informações sobre dispositivos da rede. Entretanto, a utilização geral do YANG é um processo não-trivial [Bondkovskii *et al.* 2016].

Embora reconhecendo a qualidade do *OpenDaylight*, grande parte das atividades relacionadas à instalação de regras de controle da rede deve ser realizada de forma manual, por meio da formulação de arquivos XML que serão repassados ao controlador. Basicamente, a criação de regras de controle da rede é feita pelo gerente usando comandos e instruções específicos, conforme visto em [OpenDaylight 2017a]. Além disso, o uso destes comandos depende da instalação prévia de algumas funcionalidades (*features*) em um terminal de linha de comando [OpenDaylight 2017].

2.3.2 Princípios Arquiteturais REST (*Representational State Transfer*)

Nesta seção, será apresentado um conjunto de princípios arquiteturais prevalente na concepção de arquiteturas orientadas a serviços (SOA, do inglês *Service-oriented Architectures*): **REST**, além de suas características e especificidades.

O REST (do inglês *Representational State Transfer*) consiste de um estilo arquitetural voltado para o projeto e desenvolvimento aplicações embasadas em SOA[2]. O termo REST foi criado por *Roy Fielding* e publicado em sua tese de doutorado [Fielding 2000], no ano de 2000. Desde então, este interessante conjunto de princípios tem servido como uma base fundamental para a implementação de diversas APIs públicas, voltadas para desenvolvedores de aplicações, tais como: *Google+* API [Google Inc. 2016], *Facebook* API [Facebook Inc. 2016], *Twitter* API [Twitter Inc. 2016] e *Instagram* API [Instagram Inc. 2016]. Adicionalmente, o REST vem sendo empregado em muitos estudos recentes levados a efeito na academia (vide Capítulo 3), é considerado o estilo padrão para estruturação de interfaces *northbound* [Salman *et al.* 2016] e a maioria dos projetos de controladores SDN modernos aderem aos princípios REST.

Em particular, o REST estabelece os seguintes princípios fundamentais:

- **Recursos endereçáveis:** a principal abstração de serviços baseados em REST é denominada de recurso, o qual deve ser identificado e endereçado, de forma única e inequívoca, por meio de uma URI (do inglês *Uniform Resource Identifier*). Além disso, recursos podem representar itens individuais, conjuntos de itens ou resultados de um processamento;

[2] Neste contexto, o termo SOA implica que desenvolvedores podem construir novas aplicações a partir de um conjunto reutilizável de serviços distribuídos.

- **Interface uniforme:** clientes de serviços REST utilizam um conjunto bem definido de operações (i.e., uma interface), para manipular todos os recursos existentes. Estas operações devem ser orientadas ao protocolo a partir do qual o serviço foi distribuído. Em outras palavras, os métodos usados pelos clientes para manipular os recursos são intrínsecos àqueles do protocolo utilizado;

- **Serviços orientados a representação:** recursos REST podem ter várias representações, para atender diferentes necessidades do cliente. Dessa forma, um mesmo recurso identificado por uma URI poderá apresentar vários formatos. Por exemplo, um cliente pode primeiro obter um recurso no formato JSON (do inglês *JavaScript Notation*) [Bray 2014] e, mais adiante, obter o mesmo recurso no formato XML (do inglês *eXtensible Markup Language*). O formato (ou tipo) dos recursos será selecionado por meio de negociação de conteúdo entre o serviço e o cliente. Portanto, uma URI não deve estar permanentemente associada (acoplada) a uma representação particular do recurso;

- **Comunicação cliente/serviço sem estado:** este princípio estabelece que dados ou quaisquer elementos de sessão não devem ser armazenados no servidor. Logo, se houver a necessidade de manter estado entre as interações, os dados de sessão devem ser mantidos no cliente e enviados ao servidor, sempre que for necessário;

- **Hipermídia como o motor de estado do aplicativo**: o acrônimo orginal HATEOAS (do inglês *Hypermedia As The Engine Of Application State*) é usado para identificar este princípio. HATEOAS sugere que, ao acessar um serviço, o cliente obtenha *links* embutidos diretamente no formato de dados recebido como resultado do processamento. Dessa forma, clientes poderão "navegar" a outros novos recursos, simplesmente seguindo tais ligações hipermídia. O servidor é o responsável por "guiar" o cliente do serviço e informá-lo, por meio de *links*, que novas transições ele poderá realizar.

Além das APIs REST mantidas por grandes corporações de TI citadas no início desta seção, a maioria dos trabalhos recentes encontrados na literatura, voltados para formulação de APIs *northbound* para redes SDN, está focada no REST. Isso acontece porque a aplicabilidade de outros padrões para especificação de APIs para serviços *Web*, como SOAP (do inglês *Simple Object Access Protocol*) [W3C 2007], tem sido fortemente questionada [Richardson,

Amundsen e Ruby 2013]. Em [Jakobsson, Mulligan e Unmehopa 2012], os autores destacam que SOAP não tem conseguido atrair uma larga parcela de desenvolvedores da comunidade *Web*. Na verdade, hoje em dia, é o projeto baseado em princípios REST que impera nos ambientes de desenvolvimento de APIs públicas [Burke 2013].

Diante deste cenário, muitos controladores SDN de classe empresarial têm considerado princípios e valores do REST na definição de interfaces de serviços apoiadas no protocolo HTTP (do inglês *Hypertext Transfer Protocol*), em particular. Entretanto, é importante frisar que alguns controladores não têm apresentado uma preocupação em aderir completamente aos princípios definidos por Fielding. Há controladores [Big Switch Networks 2013], por exemplo, que fornecem uma API *northbound* que (erroneamente) se auto-proclama "RESTful"; mas que viola preceitos fundamentais do REST. Ainda considerando o caso deste controlador citado, algumas partes da API *northbound* desenvolvida equivocadamente acoplam os tipos de dados às URIs dos recursos, ao invés de empregar o princípio da negociação de conteúdo entre as partes envolvidas na comunicação. Tal controlador SDN, portanto, terá dificuldades para gerenciar seu próprio espaço de nomes (do inglês *namespace*), como resultado do forte acoplamento entre clientes e serviços. Essa abordagem precisa ser repensada. Há que se ter em mente que a importância da inclusão de princípios REST no desenvolvimento de APIs *northbound* deve ser cultuada ao longo de todo o projeto.

Como se vê, ainda há espaço para avanços no processo de especificação de APIs *northbound* para redes SDN. É importante ressaltar, entretanto, que mesmo que o estilo arquitetural REST esteja ganhando cada vez mais adeptos, por conta da qualidade de seus princípios e valores fundamentais, isso não significa que ele esteja sendo corretamente aplicado e, principalmente, que as interfaces *northbound* REST desenvolvidas permitam ao utilizador programar a rede com facilidade.

2.4 Considerações Finais

Neste capítulo, foram analisadas as principais razões que levam os gerentes de redes SDN a enfrentarem dificuldades para implementar políticas e serviços capazes de programar o comportamento destas redes.

Discutiu-se, inicialmente, não apenas a dicotomia existente entre os paradigmas de redes

definidas por *software* (SDN) e de redes convencionais, mas, sobretudo, os benefícios trazidos pelas ideias do modelo SDN. Observou-se que, apesar dos esforços empreendidos com vistas a uma possível padronização das interfaces *northbound* para redes SDN, ainda não há, atualmente, um consenso na forma como os controladores existentes devem se comunicar com o plano de aplicações de mais alto nível. Em particular, percebe-se que estes controladores oferecem interfaces *northbound* não padronizadas e de baixo nível. Isto cria grandes dificuldades para os seus utilizadores, os quais acabam por ter de lidar com detalhes internos e linguagens de diferentes controladores, para programar a rede ou ampliar a integração imediata do código contendo estruturas de comandos existentes entre controladores distintos. Estudos de caso refletindo experiências no emprego de princípios SDN por parte de grandes corporações de TI também foram apresentados neste capítulo.

A revisão da literatura aponta que esforços devem ser empreendidos com vistas à simplificação do processo de instalação de regras de controle por parte do gerente sobre o controlador SDN. Toda a exposição deste capítulo ajuda a entender a necessidade da acomodação simplificada e automatizada de regras de controle em uma rede definida por *software*, em particular priorizando a interoperabilidade da interface, a partir de um corpo de funções uniforme, de alto nível e que possa ser integrado a outro controlador sem esforço.

Hoje em dia, APIs *northbound* de baixo nível que são oferecidas por muitos controladores modernos constituem um dos principais obstáculos à programação de uma rede, o que nos leva a concluir que não é fácil para seus utilizadores programar uma rede SDN, considerando tais implementações disponíveis. Estes continuarão a ser pouco capazes de formular e implantar regras para mudar o comportamento da rede independentemente do controlador existente, se a forma habitual de implementar interfaces *northbound* for mantida.

Neste sentido, os próximos capítulos deste trabalho serão dedicados à investigação de formas alternativas para programar uma rede de computadores, a partir de interfaces *northbound* agnósticas ao controlador. Dessa forma, um maior número de gerentes poderá ter melhores condições para criar políticas de gerência com valor de negócio para redes SDN, programar a SDN com maior facilidade e beneficiar-se, verdadeiramente, das vantagens do modelo de redes definidas por *software*.

Capítulo 3

Trabalhos Relacionados

Este capítulo apresenta os trabalhos encontrados na literatura que apresentam convergência com a presente pesquisa. Estes investigam, de forma direta ou indireta, questões relacionadas à programação de redes definidas por *software* (SDN) usando interfaces *northbound*.

3.1 Métodos Alternativos para Programar Redes SDN

Nos últimos anos, pesquisadores têm desenvolvido abordagens interessantes para implantar regras de controle em uma rede definida por *software*, a fim de programar seu comportamento.

Shin, Nam e Kim [Shin, Nam e Kim 2012] propuseram uma arquitetura SDN de referência, baseada em APIs abertas para comunicação entre o controlador SDN e os planos de dados e de aplicações. Este trabalho ratifica a tendência atual de utilizar controladores e interfaces de programação abertos, sejam elas *northbound* ou *southbound*. Além disso, os autores discutem a flexibilidade que a arquitetura proposta pode oferecer. A abordagem proposta nesta tese também possui seu foco no emprego de APIs e controladores emergentes de código-aberto. No entanto, a ausência de resultados de um estudo experimental mais aprofundado, a fim de avaliar uma instância da arquitetura de referência, limita a viabilidade da proposta.

Em [Wallner e Cannistra 2013; Wallner 2015], é apresentado um módulo (*qosPusher.py*) escrito na linguagem *Python* com uma interface *northbound* para que o gerente realize operações de QoS (do inglês *Quality of Service*) em redes SDN controladas pelo *Floodlight* [Big Switch Networks 2013]. Em particular, os autores definiram estruturas de comandos para encaminhar tráfego prioritário para filas de serviço, com auxílio da ação `enqueue` do

protocolo *OpenFlow* 1.0. Diferente da abordagem apresentada nesta tese, a interface proposta exige que gerentes elaborem e instalem (manualmente) as regras de controle na rede por meio de comandos de baixo nível, emitidos em terminal de linha de comando. Além disso, o módulo *qosPusher.py* opera somente com o protocolo *OpenFlow* 1.0 e, até o momento, não foi modificado para suportar versões mais recentes do *Floodlight* e *OpenFlow*. Os autores conduziram seus experimentos usando apenas redes SDN virtuais com o ambiente *Mininet* [Lantz, Heller e McKeown 2010], o *Open vSwitch* [Pfaff *et al.* 2009] e o *Floodlight* [Big Switch Networks 2013], para observar o comportamento e eficácia da solução proposta.

O *QoSFlow* [Ishimori *et al.* 2013] consiste de uma solução cuja meta é oferecer ao gerente flexibilidade e melhor gerenciabilidade em operações de QoS em redes SDN com o protocolo *OpenFlow* 1.0. Trata-se, em particular, de um *framework* composto por três módulos (*Traffic Shaping, Packet Schedulers e Enqueueing*) cuja interface permite ao gerente configurar qual escalonador de pacotes será utilizado para dar suporte às operações de QoS. Além de extensões de mensagens implementadas pelos autores para o protocolo *OpenFlow* versão 1.0, as funções oferecidas pelo *QoSFlow* utilizam o `dpctl`[1] para operar os dispositivos de rede *OpenFlow*. Equipamentos *TPLink* 1043ND com o *OpenWRT* [Yiakoumis, Schulz-Zander e Zhu 2011] foram usados pelos autores para condução dos experimentos de avaliação.

O *framework PolicyCop* [Bari *et al.* 2013] endereça o problema de gerenciamento simplificado de QoS (do inglês *Quality of Service*) em redes SDN. Trata-se, em particular, de um *framework* independente de fabricante, de código-aberto e que oferece uma interface capaz ajustar o comportamento de uma rede SDN controlada pelo *Floodlight* [Big Switch Networks 2013]. Nesse estudo, a interface de programação *northbound* é capaz de acomodar políticas de QoS na rede SDN e avaliar os seguintes parâmetros: (a) perda de pacotes; (b) *jitter*; (c) latência e (d) vazão da rede. Especificamente, os autores destacam os componentes *Policy Validator e Policy Enforcer* como os principais responsáveis por, respectivamente, detectar eventuais violações e manter as políticas de QoS, mas sem apresentar detalhes das arquiteturas de suas interfaces correspondentes ou como o gerente deve utilizar suas funções de controle.

A abordagem *PANE* [Ferguson *et al.* 2013] endereça o problema da interação do gerente com a rede em alto nível, por meio de uma API. Em seus estudos, além da API, os autores implementaram um controlador *OpenFlow* completo e um compilador, que recebe tuplas

[1] Um utilitário de linha de comando que permite realizar operações em um *switch OpenFlow*.

de comandos e as tranforma em instruções de baixo nível para o controlador. No *PANE*, em particular, as instruções são elaboradas pelo gerente através de tuplas de *elementos* de mais alto nível. Por exemplo, as requisições HTTP na rede podem ser representadas pela seguinte tupla: `<srcIP=*, dstIP=*, proto=TCP, srcPort=*, dstPort=80>`. Os autores definiram outros elementos (e.g., `TrafficBetween`, `CanDeny`, etc.) que permitem a escrita de tuplas mais complexas. Diferente da abordagem utilizada nesta tese, o *PANE* requer que o gerente escolha os elementos e construa todos os comandos necessários para programar a rede, o que aumenta bastante as chances de erros cometidos. Ademais, além do esforço para se elaborar um grande número de tuplas manualmente, estas podem ser mais restritas que o necessário, fazer uso ineficiente da rede ou até mesmo serem conflitantes.

O *HyperNet* [Huang e Griffioen 2013] consiste de um *framework* voltado para instanciação de ambientes de jogos multiusuários em redes SDN. Este trabalho explora a criação dinâmica de redes SDN, sob demanda, buscando atender sessões de jogos temporárias, cujas quantidades e localizações geográficas dos jogadores podem variar. Em particular, a API oferecida pelo *framework* traz a possibilidade de: (i) definir a topologia de rede; (ii) programar roteadores; (iii) estabelecer o servidor do jogo; (iv) identificar o nó central da rede, a partir das diferentes localizações dos jogadores, e (v) permitir que um usuário/jogador participe da rede SDN recém-criada. Apesar de úteis, não foram apresentados indícios no nível de controle da rede de que as funções oferecidas pela abordagem *HyperNet* são agnósticas ao controlador existente e, portanto, poderiam ser executadas em outros planos de gerência com características distintas.

Em [Gorlatch, Humernbrum e Glinka 2014], foi implementada uma interface *northbound* para que aplicações interativas de tempo real programem uma rede SDN. As aplicações planejadas para a API *northbound* desenvolvida na pesquisa são, essencialmente, jogos multiusuários. Nesse estudo, um módulo foi escrito na linguagem C++, o qual incorpora as principais funções da interface *northbound*. Os autores priorizam o controle da rede por meio de dois parâmetros principais: perda máxima de pacotes e taxa de transferência da rede. Embora reconhecendo a importância do estudo, a arquitetura da API *northbound* desenvolvida na pesquisa não foi apresentada. Outrossim, o estudo não informou qual controlador SDN foi escolhido para integração com a solução proposta e execução dos ensaios experimentais.

De natureza mais genérica é a interface *northbound* proposta por Chown *et al.* [Chown *et al.* 2014]. Especificamente, o estudo visa o desenvolvimento de uma arquitetura SDN

denominada OFERTIE, que possui uma API *northbound* inicial usada pelas aplicações para controlar o comportamento da rede. Não obstante a importância dada ao papel da interface *northbound* pelos seus autores, esta não é suficientemente detalhada, resultando em ceticismo no que se refere à qualidade dos parâmetros de controle da rede definidos na pesquisa. Ademais, nenhum estudo experimental foi levado a efeito, com o intuito de validar o funcionamento da interface *northbound* desenvolvida.

Em [Humernbrum, Glinka e Gorlatch 2014], uma interface *northbound* para programação da rede SDN foi elaborada, com o intuito que desenvolvedores de jogos multiusuários possam requisitar mudanças no comportamento geral da rede. Os resultados preliminares desta pesquisa foram posteriormente estendidos [Humernbrum, Glinka e Gorlatch 2014a]. Nestes estudos, a programação da rede é levada a efeito por meio de dois parâmetros de controle principais: perda máxima de pacotes e taxa de transferência da rede. Os autores conduziram seus experimentos usando apenas redes SDN virtuais com o ambiente *Mininet* [Lantz, Heller e McKeown 2010] e o *Open vSwitch* [Pfaff *et al.* 2009], para observar o comportamento e eficácia das interfaces *northbound* propostas. Em ambos os trabalhos, contudo, as arquiteturas das APIs *northbound* desenvolvidas não foram apresentadas, nem tampouco foram feitas recomendações sobre o que é necessário para compatibilizar as funções da interface proposta com algum controlador SDN de classe empresarial moderno.

O *RestChartModel* [Li e Chou 2011] utiliza Redes de Petri para modelar APIs *Web* baseadas em princípios REST (do inglês *Representational State Transfer*) [Fielding 2000]. O modelo apresentado na pesquisa permite verificar se todos princípios REST estão sendo considerados corretamente. O autores utilizaram o *RestChartModel* para desenvolver uma API REST simples. Contudo, os serviços desta API REST de exemplo desenvolvida não foram validados pelos autores—seja em um ambiente real ou utilizando algum suporte ferramental para simulação. A partir dos resultados desta pesquisa, Zhou *et al.* [Zhou *et al.* 2014] propuseram um *plugin*—o *SoxProxy*—capaz de receber instruções da API REST de exemplo preliminar e compatibilizá-las com o controlador SOX [Luo *et al.* 2012], o qual é mantido pela empresa *Huawei* [2]. Posteriormente, em [Zhou, Li e Chou 2014], os autores propuseram uma abordagem baseada no uso de *caches*, para melhorar o desempenho de acesso aos recursos da rede através das URIs da API REST. Apesar dos importantes desdobramentos de pesquisa

[2] <http://www.huawei.com>

realizados nos dois estudos preliminares, a API REST desenvolvida toma como foco um controlador SDN que possui inúmeras deficiências [Akyildiz *et al.* 2014], é mantido por uma única empresa, fechado e, dessa forma, pouco flexível para acomodar extensões.

A abordagem NOSIX [Yu, Wundsam e Raju 2014] descreve uma camada de abstração bastante interessante, cujo principal objetivo é garantir a portabilidade das instruções aplicadas às tabelas de fluxos de dispositivos *OpenFlow* de diferentes fabricantes. Em particular, a interface de operação desenvolvida consiste de uma linguagem de anotação, cujas estruturas são traduzidas em instruções específicas do equipamento-alvo por meio de um *driver* fornecido pelo próprio fabricante. Além de buscar padronizar o processamento de tabelas de fluxo em redes SDN, o qual será realizado em mais alto nível, a linguagem de anotação proposta pela pesquisa evita que o gerente tenha de lidar com instruções e outros detalhes de implementação peculiares ao fabricante do equipamento existente na SDN. Para fins de avaliação, autores reportaram a utilização de dispositivos compatíveis com *OpenFlow* na versão 1.0, mas sem informar qual controlador SDN foi selecionado para execução dos ensaios experimentais.

Em [Casey, Sutton e Sprintson 2014], os autores propuseram uma interface denominada *tinyNBI* produzida a partir de cinco diferentes versões do protocolo *OpenFlow*. Em particular, as funções da API foram escritas na linguagem C e são totalmente agnósticas à versão *OpenFlow* suportada pelo equipamento de rede. Os autores colocam que a interface *tinyNBI* busca unificar e padronizar abstrações e outras funcionalidades derivadas de versões *OpenFlow* heterogêneas, tendo em vista esconder detalhes de implementação dos protocolos, reduzir a variabilidade e também a complexidade para programar o dispositivo de uma rede SDN.

Palma *et al.* [Palma *et al.* 2014] apresentam uma interface *northbound*, chamada *Queue-Pusher*, voltada para o controlador *Floodlight* [Big Switch Networks 2013]. A abordagem proposta nesta pesquisa inspira-se em princípios REST [Fielding 2000] e utiliza o protocolo OVSDB (do inglês *Open vSwitch Database Management Protocol*) [Pfaff e Davie 2013] para o gerenciamento em alto nível de filas de QoS em portas do *Open vSwitch* [Pfaff *et al.* 2009]. Neste trabalho, uma topologia de rede SDN virtual foi usada para realização dos experimentos de simulação. Apesar do foco em princípios REST, infelizmente, os autores não informaram quais os formatos das URIs REST definidas, muito menos os parâmetros que devem ser manipulados pelo gerente para criar, remover e modificar filas de serviços prioritários na rede.

Considerando ambientes de computação na nuvem, o trabalho de Lin *et al.* [Lin *et al.* 2014] apresentou um módulo para controle da rede, chamado *SDI Manager*. Este possui uma interface *northbound*, que permite às aplicações de *software* controlar o comportamento da rede SDN. Em particular, uma arquitetura SDN bastante interessante foi desenvolvida, que inclui o controlador *Ryu* [Nippon Telegraph and Telephone Corporation 2012] atuando como um *backend* (estrutura de retaguarda) para o *OpenStack* [OpenStack 2014]. A interface *northbound* permite controlar determinados aspectos da rede, tais como: criação e remoção de redes virtuais completas e delegação de controle sobre uma determinada rede virtual. Entretanto, cumpre ressaltar que, apesar de pertinente, este trabalho não apresentou quaisquer informações quanto aos protótipos das funções desenvolvidas, à forma de integração com o *Ryu* ou ainda como o corpo de funções deve ser utilizado pelo gerente para controlar a rede.

Kondwilkar *et al.* [Kondwilkar *et al.* 2015] apresentam uma plataforma de gerência de redes desenvolvida para o controlador *OpenDaylight* [OpenDaylight 2013]. A solução proposta oferece uma API *northbound* baseada no padrão REST [Fielding 2000], para que as aplicações controlem o comportamento da rede. Em particular, um *script* escrito na linguagem de programação *Python* foi desenvolvido pelos autores, que contém toda a lógica de funcionamento da API *northbound*. Uma arquitetura bastante genérica da API REST é apresentada neste trabalho. Além do controlador *OpenDaylight*, os autores conduziram seus experimentos usando uma rede SDN virtual simulada no ambiente *Mininet* [Lantz, Heller e McKeown 2010], para observar o comportamento da solução proposta. Infelizmente, ao contrário da abordagem proposta nesta tese, não foram informados os formatos das URIs que acessam recursos REST, muito menos detalhes acerca dos principais parâmetros de controle que devem ser manipulados pelo gerente para efetivamente programar a rede SDN.

Em um estudo mais recente, Gorlatch e Humernbrum [Gorlatch e Humernbrum 2015] buscaram definir uma interface *northbound* mais focada na perspectiva do desenvolvedor de aplicações para redes SDN e suas habilidades. Neste estudo, em vez de efetuar o controle da rede por meio de métricas tradicionais, como *jitter* ou perda de pacotes, os autores propuseram a programação da rede por meio de uma métrica de controle de mais alto nível—tempo de resposta—para que desenvolvedores de aplicações interativas de tempo real possam ajustar o comportamento da rede, em tempo de execução. Para execução dos experimentos, os autores utilizaram uma topologia SDN contendo um controlador desenvolvido pelos próprios autores,

dispositivos *Open vSwitch* [Pfaff *et al.* 2009] e um servidor de jogos. Diferentemente da abordagem proposta nesta tese de doutorado, os autores não informaram detalhes quanto ao conjunto de funções (e seus protótipos) disponíveis na API *northbound* desenvolvida.

Banse e Rangarajan [Banse e Rangarajan 2015] propuseram uma interface *northbound* baseada em princípios REST [Fielding 2000] e com foco principal na segurança. Chamada de *Controller Service*, a API foi implementada usando a linguagem *Java* e concede acesso às informações da rede (e.g., estatísticas de tráfego, eventos gerados, etc.) apenas àquelas aplicações previamente credenciadas e confiáveis. Por conta de um sistema de permissões especial, as aplicações ficam impedidas de executar atividades maliciosas que causem danos à rede SDN. A interface apresentada pelos autores está integrada ao controlador *Floodlight* [Big Switch Networks 2013], da mesma forma que a abordagem proposta no presente trabalho. Na pesquisa, além deste controlador, equipamentos modelo HP 3500-24 foram utilizados para condução dos experimentos. Contudo, infelizmente, os autores não informaram quais os formatos das URIs utilizadas pelos usuários da API para acessar recursos da rede.

À respeito da programação de redes SDN em ambientes MMOG (do inglês *Massive Multiplayer Online Games*), a abordagem apresentada por Humernbrum *et al.* [Humernbrum *et al.* 2015] explora o controle da rede por meio da integração entre o *framework Real-Time Framework* (RTF) com o *Shark 3D* [Spinor-GmbH 2015]. Em seus estudos, os autores implementaram uma interface *northbound* capaz de controlar a rede SDN por meio de dois parâmetros de controle principais: vazão mínima e tempo máximo de resposta da aplicação. Embora reconhecendo que a interface *northbound* desenvolvida permita modificar o comportamento da rede por meio de métricas mais adequadas ao desenvolvedor de jogos (do que *jitter*, por exemplo), esta não foi suficientemente detalhada.

Semelhante à pesquisa preliminar desenvolvida pelos autores Palma *et al.* [Palma *et al.* 2014], um estudo mais recente [Caba e Soler 2015] descreveu a proposta de uma interface, chamada *QoS Config API*. Trata-se, em particular, de um *plugin* desenvolvido para o controlador *Floodlight* [Big Switch Networks 2013] e que usa funcionalidades do protocolo OVSDB (do inglês *Open vSwitch Database Management Protocol*) [Pfaff e Davie 2013] para endereçar o problema do gerenciamento em alto nível de filas de serviço prioritário em redes SDN. As funções oferecidas pela *QoS Config API* permitem ao gerente criar, remover e listar as filas de serviço prioritário em portas do *Open vSwitch* [Pfaff *et al.* 2009]. Em seus

experimentos, os autores usaram apenas dispositivos *Open vSwitch* operando com o protocolo *OpenFlow* na versão 1.0. Os autores descatam que as funções da API seguem princípios REST [Fielding 2000], mas não apresentaram os formatos das URIs REST e os argumentos que devem ser manipulados pelo usuário da API para configurar filas de serviço na rede. Além disso, a solução desenvolvida pelos autores só é bem aplicada em redes SDN onde todos os dispositivos do plano de dados ofereçam suporte nativo ao protocolo OVSDB—o que, muitas vezes, não condiz com a realidade da maioria dos ambientes de rede SDN atuais.

A abordagem *Ryuo* [Zhang *et al.* 2015] explora a otimização do processo de tradução das instruções de controle da rede. Em particular, após receber as instruções das aplicações do negócio, a API *northbound* oferecida pelo *Ryuo* busca reduzir o número de instruções de controle (de baixo nível) necessárias para orquestrar a rede. Nesse estudo, os autores selecionaram o controlador *Ryu* [Nippon Telegraph and Telephone Corporation 2012] para integração com a solução proposta. Não obstante a relevância da pesquisa, infelizmente, não foram informados pelos autores os protótipos das funções desenvolvidas e os principais parâmetros de controle devem ser manipulados pelos usuários da API do *Ryuo* para poderem usufruir da otimização de instruções de controle e redução do volume de tráfego na SDN.

Em [Sieber *et al.* 2015], uma camada de abstração denominada NSAL foi elaborada com o intuito de promover o gerenciamento de QoS em alto nível em ambientes de rede SDN. Em particular, NSAL está situada no topo do plano de controle de uma rede SDN e define um conjunto de abstrações e funcionalidades integrado ao controlador *OpenDaylight* [OpenDaylight 2013]. Na pesquisa, além deste controlador, a plataforma *StableNet* [Infosim 2015] e topologias SDN de produção contendo equipamentos NEC PF5240 e *Cisco Catalyst* 3650 foram utilizados para condução dos experimentos. Os autores citam que NSAL foi capaz de acomodar regras de controle na rede SDN, tendo em vista atender às demandas de QoS de aplicações VOIP (do inglês *Voice Over Internet Protocol*). Contudo, infelizmente, as funções que devem ser acessadas pelos usuários da solução proposta bem como a integração entre NSAL e o controlador *OpenDaylight* não foram suficientemente detalhadas.

A pesquisa de Yan *et al.* [Yan *et al.* 2015] desenvolveu uma aplicação—*HiQoS*—que suprime detalhes internos de funcionamento do controlador SDN e permite ao gerente alocar fluxos de tráfego multimídia às filas de prioridades, efetuar o controle de reserva de largura de banda e computar a melhor rota de comunicação entre pontos remotos da rede com mais

facilidade. Em seus estudos, os autores selecionaram o *Floodlight* [Big Switch Networks 2013] como o único controlador-alvo e os experimentos foram conduzidos em redes SDN virtuais instanciadas no *Mininet* [Lantz, Heller e McKeown 2010] com dispositivos *Open vSwitch* [Pfaff *et al.* 2009]. Embora reconhecendo a importância da pesquisa, o principal fator de limitação da proposta é que *HiQoS* instaura regras de controle diretamente nos dispositivos da rede SDN usando instruções do protocolo OVSDB [Pfaff e Davie 2013], em vez de integrar-se com a interface do *Floodlight*. O resultado desta limitação é que, por não utilizar funções nativas do controlador, o *HiQoS*, por exemplo, renuncia os processos de verificação e garantia de integridade de todas as instruções que serão instaladas na rede, além de outras tarefas automáticas desempenhadas pelo *Floodlight* com o objetivo de aumentar a produtividade, a eficiência e a facilidade de programar a rede. Por fim, os autores destacam o algoritmo de computação de rotas desenvolvido na pesquisa, mas, infelizmente, não apresentam uma discussão mais aprofundada sobre os formatos das principais funções oferecidas ao gerente para operar os dispositivos da rede ou como este profissional deve efetivamente utilizá-las.

Em [Xu, Chen e Qian 2015], é apresentado um *framework* desenvolvido para o controlador *Ryu* [Nippon Telegraph and Telephone Corporation 2012] e que toma como foco o provisionamento simplificado de QoS em redes SDN. A arquitetura geral da solução proposta, que contém uma API *northbound*, é destacada pelos autores na pesquisa. Em linhas gerais, o *framework* busca abstrair o processo de configuração de filas de serviços prioritários em dispositivos *OpenFlow*, além de facilitar a alocação dinâmica de fluxos de tráfego às filas existentes. Além do *Ryu*, os autores conduziram seus experimentos usando topologias de redes SDN reais e virtuais simuladas no ambiente *Mininet* [Lantz, Heller e McKeown 2010] com dispositivos *Open vSwitch* [Pfaff *et al.* 2009], para validar o funcionamento do *framework* proposto. Não obstante a importância dada ao papel do *framework* pelos seus autores, sua interface de uso não foi suficientemente detalhada. Por exemplo, não foram apresentados detalhes quanto aos protótipos das funções (e seus parâmetros) oferecidos ao gerente ou ainda exemplos de como o corpo de funções deve ser efetivamente utilizado para programar a rede.

A abordagem RAPTOR [Rivera, Fei e Griffioen 2016] consiste de uma interface *northbound* baseada em princípios REST (do inglês *Representational State Transfer*) [Fielding 2000] que procura endereçar o problema da ausência de portabilidade das operações de controle realizadas entre controladores SDN distintos. Especificamente, a interface é um protótipo

inicial contendo apenas três funções (`GET/(static)flows`, `DELETE/flows` e `GET/switches`) desenvolvidas para operar os controladores *Ryu* [Nippon Telegraph and Telephone Corporation 2012] e *Floodlight* [Big Switch Networks 2013]. Na pesquisa, os autores não implementaram funções capazes de acomodar (instalar) regras de controle na rede. Em vez disso, esta tarefa foi delegada a um módulo externo, chamado *Flow Handler Module*, que é parte integrante do GENI (do inglês *Global Environment for Network Innovations*) [Berman *et al.* 2014]. Esta plataforma é voltada principalmente para instanciação dinâmica de ambientes de testes (*testbeds*) para experimentação em SDN virtuais. Os autores afirmam que a interface proposta permite às aplicações programar a rede SDN independentemente do controlador existente. Entretanto, ao se optar por RAPTOR, alguns pontos devem ser considerados: as URIs REST permitem apenas efetuar a listagem e remoção das regras de controle existentes na rede; a instalação de instruções na rede depende inteiramente da adoção da plataforma GENI—o que nem sempre pode ser garantido em cenários de redes SDN reais com características distintas das que foram apresentadas no cenário investigado pelos autores. Na verdade, ao afirmar que as funções RAPTOR são agnósticas ao controlador existente, considera-se uma visão otimista, posto que não são dados indícios de que elas realmente podem ser usadas para programar a rede sem utilizar funcionalidades do GENI. Por outro lado, na abordagem proposta nesta tese, tem-se funções que: (a) comunicam-se apenas com as interfaces nativas dos controladores; (b) apóiam-se em um modelo de dados JSON unificado; (c) não pressupõem a adoção de outra plataforma externa para programar a rede; e (d) oferecem ao gerente não apenas funções de alto nível para listagem e remoção, mas, sobretudo, instalação de instruções de controle tendo em vista programar o comportamento da SDN.

Similar ao estudo descrito em [Shin, Nam e Kim 2012], os autores Belzarena *et al.* [Belzarena *et al.* 2016] propuseram uma arquitetura SDN de referência, que contém uma interface *northbound* voltada para aplicações que precisam programar a rede SDN tendo em vista atender determinadas demandas de QoS. Nesta pesquisa, apesar da importância e o papel da interface serem discutidos pelos seus autores, esta não é suficientemente detalhada, resultando em ceticismo no que se refere à qualidade dos parâmetros de controle da rede definidos na pesquisa. Ademais, nenhum estudo experimental foi levado a efeito, com o intuito de validar o funcionamento da arquitetura e interface *northbound* propostas.

Os autores Layeghy, Pakzad e Portmann [Layeghy, Pakzad e Portmann 2016] apresentaram

uma interface *northbound*, denominada SCOR, voltada para o controlador POX [McCauley 2012]. Em particular, a solução proposta consiste de uma linguagem declarativa e um conjunto de preditores de alto nível que permitem ao gerente de redes acomodar requisitos de QoS na rede. Para formular regras de QoS, SCOR exige que o gerente corretamente selecione os preditores e escreva (manualmente) as instruções que serão instaladas na rede. Após este processo, um tradutor compilará as instruções elaboradas e as repassará ao controlador POX. Embora reconhecendo a importância da pesquisa, o principal fator de limitação da proposta é que SCOR não permite que os preditores sejam reutilizados em outros controladores. Além disso, até o momento da escrita deste documento, o controlador escolhido pelos autores para integração com SCOR oferece suporte apenas ao protocolo *OpenFlow* na versão 1.0 e possui deficiências, principalmente aquelas relacionadas à flexibilidade, modularidade, documentação e capacidade transacional, conforme se discute em [Salman *et al.* 2016]. Para validar o funcionamento da interface *northbound* proposta, os autores conduziram seus experimentos por meio de simulações em uma topologia de rede SDN virtual.

Em [Pham e Hoang 2016], é apresentada uma interface *northbound* baseada na intenção (*intent-based*) para que usuários e aplicações possam programar a rede SDN em alto nível. Com uma interface baseada na intenção, segundo os autores, pode-se descrever os requisitos que serão acomodados na rede usando estruturas de alto nível muito próximas da linguagem natural. A partir daí, o controlador irá processar as estruturas elaboradas e produzir todas as instruções de controle (de baixo nível) necessárias para programar a SDN. Em seus estudos, os autores selecionaram o ONOS [Berde *et al.* 2014] como o único controlador-alvo para integração com a solução proposta. A interface desenvolvida na pesquisa é um protótipo inicial. Os autores destacam os princípios que orientam a concepção de interfaces baseadas na intenção, mas, infelizmente, não apresentam uma discussão mais aprofundada sobre as estruturas de linguagem oferecidas para operar os dispositivos da rede, como se dá o processo de tradução destas estruturas em instruções de controle de baixo nível ou como elas devem ser efetivamente utilizadas. Experimentos foram conduzidos apenas em redes SDN virtuais instanciadas no ambiente *Mininet* [Lantz, Heller e McKeown 2010] com o *Open vSwitch* [Pfaff *et al.* 2009], para observar o comportamento e eficácia da interface proposta.

A pesquisa de Sieber *et al.* [Sieber *et al.* 2016] descreveu uma arquitetura de referência e o protótipo inicial de uma camada para o gerenciamento de redes tendo em vista o controle

padronizado de dispositivos *OpenFlow* e legados usando abstrações de alto nível. Além da operação e monitoramento de equipamentos de rede heterogêneos, a solução proposta é capaz de verificar e acomodar requisitos de QoS em ambientes de rede SDN. Os autores afirmam que a arquitetura desenvolvida possui uma interface *northbound* baseada em princípios REST [Fielding 2000], mas sem apresentar os formatos das URIs REST definidas e os parâmetros que devem ser manipulados pelo gerente para modificar o comportamento da rede.

A abordagem *SemSDN* [Chen e Wu 2017] explora o desenvolvimento de uma interface *northbound* inspirada em princípios de *Web* Semântica. Especificamente, a solução proposta consiste de uma linguagem declarativa baseada em SPARQL [W3C 2013] e um conjunto de funções para gerenciamento de recursos da rede SDN. As funções da interface *SemSDN* são acessadas pelo gerente por meio de *Servlets Java*. Os autores desenvolveram cinco funções para: (a) identificar *loops* na rede, (b) efetuar a computação de rotas, (c) identificar a topologia e recolher informações de QoS existentes na rede, (d) coletar funcionalidades de QoS disponíveis no equipamento *OpenFlow*, e (e) imprimir gráfico da topologia de rede. O *Floodlight* [Big Switch Networks 2013] foi o único controlador-alvo escolhido para integração com a interface *SemSDN* proposta. Além disso, experimentos de simulação foram conduzidos em topologias SDN virtuais contendo dispositivos *Open vSwitch* [Pfaff *et al.* 2009].

Em pesquisa recente levada a efeito em [Layeghy, Pakzad e Portmann 2017], os autores aprimoraram os resultados preliminares apresentados em [Layeghy, Pakzad e Portmann 2016]. Particularmente, 02 (dois) novos preditores para acomodação de instruções de QoS na rede foram incorporados à interface *northbound* SCOR. Não obstante as deficiências relacionadas à flexibilidade, modularidade, documentação e capacidade transacional, discutidas em [Salman *et al.* 2016], os autores mantiveram o POX [McCauley 2012] como o único controlador integrado à interface proposta. Embora reconhecendo os importantes desdobramentos obtidos na pesquisa, SCOR ainda não oferece nenhum mecanismo de automatização para instalação de instruções na rede. Na verdade, SCOR exige que o gerente selecione os preditores e construa (manualmente) todos os comandos necessários para programar a rede, o que aumenta as chances de erros cometidos. Ainda, não é permitido reutilizar os preditores em outro controlador. Para validar o funcionamento da interface com os novos preditores, os autores conduziram seus experimentos por meio de simulações em uma topologia SDN virtual.

Em [Pan *et al.* 2017], é apresentada uma arquitetura interessante, denominada *OpenSched*, para o provisionamento simplificado de QoS em redes SDN. Esta arquitetura contém uma interface *northbound* voltada para o controlador ONOS [Berde *et al.* 2014] e um componente (*QoS Policy Parser*) capaz de traduzir as requisições recebidas da interface em instruções de baixo nível para posterior instalação nos equipamentos da rede. A interface desenvolvida é um protótipo inicial contendo 03 (três) funções responsáveis por, respectivamente: (a) realizar uma conexão e registrar um dispositivo de rede, (b) consultar o estado e colher informações sobre a porta do equipamento de rede, e (c) obter a política de QoS do componente (*QoS Policy Parser*) e acomodá-la na SDN. A invocação das funções é bastante semelhante à chamada de um construtor na linguagem C++. Para condução dos experimentos, os autores utilizaram uma topologia SDN virtual com dispositivos *Open vSwitch* [Pfaff *et al.* 2009].

O MISSIn [Neto *et al.* 2017] consiste de uma ferramenta de apoio para a orquestração de infraestruturas SDN que trabalha de forma interativa e dinâmica. Em particular, os autores citam que a solução permite abstrair a complexidade do gerenciamento da rede, além de fornecer a autonomia para usar qualquer controlador para gerenciar a infraestrutura SDN. A arquitetura da proposta do sistema MISSIn é formada pelos principais componentes: (a) *MISSIn Logic Unit*: onde o operador pode acessar um mapa da infraestrutura de rede; (b) *GUI*: um sistema *Web* onde é possível interagir com os dispositivos da rede; (c) *Resource Manager*: composto por mecanismos de controle de recursos; e (d) *MISSIn Server*: componente responsável pela intermediação entre a GUI e o controlador. Especificamente, os autores destacam os sub-componentes *Topology Manager* e *QoS Manager* como os principais responsáveis por, respectivamente, obter informações sobre a composição e estado da rede com interação entre operador e os dispositivos; e a criação e manutenção de políticas de QoS com capacidades de reservas de recursos. Apesar no foco no controle da rede em alto nível e integração com múltiplos controladores, infelizmente, os autores não apresentaram os protótipos das funções de controle desenvolvidas, muito menos que parâmetros devem ser manipulados pelo gerente para criar, remover e modificar instruções de QoS na rede. Ao contrário da abordagem proposta nesta tese, os autores não forneceram detalhes quanto aos modelos de dados utilizados na compatibilização do MISSIn às interfaces nativas dos diferentes controladores. No MISSIn, o componente agente responsável pela comunicação com o controlador ainda encontra-se em fase experimental e futuros testes ainda serão realizados em

ambiente simulado com o Mininet [Lantz, Heller e McKeown 2010]. Por fim, nenhum estudo experimental foi levado a efeito na pesquisa, com o intuito de validar o funcionamento da interface de controle do MISSIn, o que limita a viabilidade da proposta.

Para ilustrar os trabalhos anteriormente descritos, pode-se observar, na Tabela 3.1 (pág. 50), um quadro comparativo, que contém uma descrição resumida da solução proposta, informações quanto à interface para programação da rede desenvolvida, o ambiente de avaliação definido, o controlador selecionado pelo estudo (quando informado) e, por fim, uma indicação que informa se o mesmo corpo de funções da interface pode prontamente comunicar-se com outro controlador SDN de classe empresarial distinto, isto é, se a interface permite a interoperabilidade de todas as suas funcionalidades existentes.

Tabela 3.1: Quadro comparativo das abordagens correlatas

	Trabalho	Descrição da solução	Detalhamento da interface	Ambiente de avaliação	Controlador(es)	Interoperabilidade
1	[Shin, Nam e Kim 2012]	Uma arquitetura SDN de referência e uma interface *northbound* para programar a rede SDN	A interface não é suficientemente detalhada	Não informado	Não informado	Não
2	[Wallner e Cannistra 2013] [Wallner 2015]	Um módulo escrito em *Python* (***qosPusher.py***) com uma interface para dar suporte às operações de QoS em redes SDN com protocolo *OpenFlow* 1.0	Gerente deve formular e aplicar comandos em terminal de linha de comando para instalar/remover políticas de QoS na SDN. ***qosPusher.py*** provê estruturas de comandos para encaminhar tráfego marcado para filas de serviço prioritário, usando a propriedade *enqueue action* do protocolo *OpenFlow* versão 1.0	Simulações no ambiente *Mininet* e topologia SDN virtual com *Open vSwitch*	*Floodlight*	Não
3	[Ishimori *et al.* 2013]	***QoSFlow*** consiste de uma solução para permitir a mudança de escalonador em uma rede SDN com *OpenFlow* versão 1.0 visando flexibilidade e melhor controle de operações de QoS	As funções do ***QoSFlow*** usam o utilitário `dpctl` e um algoritmo para operar o dispositivo de rede e escolher o escalonador a ser aplicado, a partir de extensões de mensagens propostas para o protocolo *OpenFlow* versão 1.0	Testes conduzidos em topologias SDN compostas de 1, 2 e 3 equipamentos *TPLink* 1043ND com *OpenWRT* embarcado	Não informado	Não
4	[Bari *et al.* 2013]	***PolicyCop*** é um *framework* para o gerenciamento simplificado de operações de QoS em redes SDN controladas pelo *Floodlight*	Deve-se utilizar os componentes *Policy Validator e Policy Enforcer* para detectar violações, estabelecer políticas de QoS na SDN e observar parâmetros da rede, como: perda de pacotes, *jitter*, taxa de transmissão, latência e número de falhas por dispositivo; contudo, as sintaxes de uso das funções não foram apresentadas	Topologia SDN com 5 *switches Open vSwitch* e 4 nós	*Floodlight*	Não
5	[Ferguson *et al.* 2013]	**PANE** consiste de um compilador, um controlador SDN e uma interface *northbound* para interação e programação da rede SDN em alto nível	**PANE** exige que o gerente construa manualmente o algoritmo para programar a SDN usando tuplas de elementos definidos na pesquisa (e.g, *TrafficBetween*, *CanDeny*, etc.). Um compilador recebe as tuplas elaboradas e as traduz em instruções compatíveis com a interface do controlador	Topologias SDN reais e virtuais usando *Mininet*, *Open vSwitch* e equipamentos *TP-Link* WR-1043ND	Controlador desenvolvido pelos autores	Não

Continua na próxima página

Tabela 3.1 – continuação da página anterior

	Trabalho	Descrição da solução	Detalhamento da interface	Ambiente de avaliação	Controlador(es)	Interoperabilidade
6	[Huang e Griffioen 2013]	***HyperNet*** é um *framework* voltado para estabelecimento de sessões de jogos multiusuários em ambientes de redes SDN	A interface proposta permite instanciar redes SDN completas sob demanda para atender sessões de jogos temporárias. ***HyperNet*** oferece funções para definir a topologia, localizar roteadores, escabelecer o servidor de jogo, identificar o nó central da rede, registrar o *host* do jogador, dentre outras funcionalidades	Simulação com SDN virtuais no ambiente ProtoGENI	Não informado	Não
7	[Gorlatch, Humernbrum e Glinka 2014]	***SDN Module*** consiste de uma interface *northbound* que permite às aplicações interativas de tempo real programar parâmetros de QoS na rede SDN	A interface é materializada como uma biblioteca escrita na linguagem C++. Em particular, o gerente poderá configurar dois parâmetros principais: a perda máxima de pacotes e a taxa de transferência da SDN. As sintaxes de uso das funções e seus protótipos não foram detalhadas	Não informado	Não informado	Não
8	[Chown *et al.* 2014]	**OFERTIE** consiste de uma arquitetura SDN completa contendo uma interface capaz de programar a rede SDN em alto nível	A interface proposta na pesquisa não é suficientemente detalhada. Autores não discutiram os protótipos das funções, a compatibilização com a interface nativa do controlador ou quais os parâmetros disponíveis para controle da rede	Não informado	Não informado	Não
9	[Humernbrum, Glinka e Gorlatch 2014] [Humernbrum, Glinka e Gorlatch 2014a]	Proposta de uma interface *northbound* para programar a rede SDN a partir de demandas de sessões de jogos multiusuários	Continuação do trabalho [Gorlatch, Humernbrum e Glinka 2014]. A interface proposta permite configurar dois parâmetros: a perda máxima de pacotes e a taxa de transferência da rede. Em ambas as pesquisas, as sintaxes de uso das funções e seus argumentos não foram detalhados	Topologias SDN instanciadas no *Mininet* contendo 6 nós e 3 *switches Open vSwitch*	Controlador de referência nativo do *Mininet*	Não
10	[Zhou *et al.* 2014] [Zhou, Li e Chou 2014]	***SoxProxy*** é um *plugin* para o controlador SOX que oferece uma interface baseada em princípios REST	Continuação do trabalho [Li e Chou 2011]. A interface REST proposta usa recursos da linguagem *Python* para interagir com API nativa do SOX. ***SoxProxy*** permite ao gerente interagir com o SOX tendo em vista obter informações de dispositivos do plano de dados SDN	Topologia SDN real composta de 2 nós e 1 servidor *Huawei Tecal* RH2285	SOX	Não

Continua na próxima página

Tabela 3.1 – continuação da página anterior

	Trabalho	Descrição da solução	Detalhamento da interface	Ambiente de avaliação	Controlador(es)	Interoperabilidade
11	[Yu, Wundsam e Raju 2014]	**NOSIX** representa uma camada de abstração uniforme, que visa garantir a portabilidade de operações realizadas sobre as tabelas de fluxos de equipamentos de rede de diferentes fabricantes	Trata-se de uma linguagem de anotação, cujas estruturas de alto nível são traduzidas em instruções específicas do dispositivo de rede por meio de *driver* fornecido pelo fabricante. **NOSIX** busca padronizar o processamento das instruções aplicadas às tabelas de fluxos de dispositivos *OpenFlow* heterogêneos. Em vez de lidar com detalhes de operação peculiares ao fabricante do equipamento, o gerente utiliza as estruturas da linguagem para operar as chamadas *tabelas virtuais*	Simulação usa equipamentos compatíveis com o protocolo *OpenFlow* versão 1.0 (modelo não informado)	Não informado	Não
12	[Casey, Sutton e Sprintson 2014]	***tinyNBI*** consiste de uma interface de baixo nível produzida a partir de 5 diferentes versões do protocolo *OpenFlow*	***tinyNBI*** oferece um conjunto de estruturas e funções escritas na linguagem C totalmente agnóstico à versão *OpenFlow* suportada pelo *switch*. Gerente deve escrever código em C para acomodar instruções na rede	Não informado	*Freeflow*	Não
13	[Palma *et al.* 2014]	***QueuePusher*** é uma solução que visa o gerenciamento em alto nível de filas de QoS em portas do *Open vSwitch* em redes SDN controladas pelo *Floodlight*	A interface de ***QueuePusher*** inspira-se em princípios REST e usa funcionalidades do protocolo OVSDB para comunicar-se com *Floodlight*; 2 funções foram implementadas: criação e remoção de filas de QoS em *swiches Open vSwitch*. Contudo, os formatos das URIs REST definidas não foram apresentados	Simulação em topologia SDN virtual com *Open vSwitch*	*Floodlight*	Não
14	[Lin *et al.* 2014]	***SDI Manager*** é um módulo para controle em alto nível da rede SDN contendo uma interface *northbound*	A interface proposta contém 2 funções primárias: criação e remoção de redes SDN virtuais completas e delegação de controle sobre uma rede virtual; informações quanto aos protótipos das funções desenvolvidas ou à forma de integração com o *Ryu* não foram apresentadas	Simulações conduzidas no ambiente SAVI	*Ryu*	Não

Continua na próxima página

Tabela 3.1 – continuação da página anterior

	Trabalho	Descrição da solução	Detalhamento da interface	Ambiente de avaliação	Controlador(es)	Interoperabilidade
15	[Kondwilkar *et al.* 2015]	Proposta de uma plataforma de gerência de redes SDN voltada para o controlador *OpenDaylight* com uma interface *northbound*	Autores citam que a interface proposta adere a princípios REST. Porém, um *script* em *Python* contém o funcionamento da API. Uma arquitetura preliminar bastante genérica da API é discutida; mas, as sintaxes, os principais parâmetros de controle e exemplos de uso das URIs REST não foram detalhados	Topologia SDN virtual, com 4 nós e 4 *switches*, instanciada no *Mininet*	*OpenDaylight*	Não
16	[Gorlatch e Humernbrum 2015]	Uma interface *northbound* para que aplicações interativas de tempo real (e.g., jogos) possam acomodar requisitos de QoS na rede SDN	Continuação dos trabalhos [Humernbrum, Glinka e Gorlatch 2014] e [Humernbrum, Glinka e Gorlatch 2014a]. A interface foi aprimorada e permite gerenciar o tempo de resposta máximo da aplicação. As sintaxes de uso das funções e seus argumentos não foram detalhados	Topologia SDN composta de 3 dispositivos *Open vSwitch*, um servidor de jogo real e vários nós (os jogadores)	Protótipo inicial de um controlador definido pelos autores da pesquisa	Não
17	[Banse e Rangarajan 2015]	***Controller Service*** consiste de uma interface *northbound* para o controlador *Floodlight* com foco na segurança e no controle de acesso aos recursos da rede SDN	A interface é um *Web service* escrito em *Java* e concede acesso às informações da rede (e.g., estatísticas de tráfego, eventos gerados, etc.) apenas às aplicações previamente credenciadas e confiáveis. Há funções para permitir/restringir acesso aos recursos da SDN por meio de autenticação e estabelecimento de conexões criptografadas com o controlador	Topologia SDN composta de equipamentos reais (HP 3500-24) e virtuais (*Open vSwitch*)	*Floodlight*	Não
18	[Humernbrum *et al.* 2015]	Uma interface para programação de redes SDN utilizando o *Real-Time Framework (RTF)* e o ambiente *Shark 3D* para atender demandas de jogos *online*	A interface proposta é capaz de programar a SDN por meio de dois parâmetros de controle principais: vazão mínima e tempo máximo de resposta da aplicação (jogo). O conjunto de funções desenvolvido não foi detalhado pelos autores	Simulação conduzida no FIBRE em topologia composta por 4 *switches* e 4 nós clientes (jogadores)	Não informado	Não

Continua na próxima página

Tabela 3.1 – continuação da página anterior

	Trabalho	Descrição da solução	Detalhamento da interface	Ambiente de avaliação	Controlador(es)	Interoperabilidade
19	[Caba e Soler 2015]	A ***QoS Config API*** é uma interface inspirada em princípios REST desenvolvida para o controlador *Floodlight* tendo em vista simplificar a configuração de filas de prioridade em portas do *Open vSwitch*	Trata-se de um *plugin* compatível com o *Floodlight* e cujo corpo de funções utiliza funções do protocolo OVSDB para o gerenciamento (criação, remoção e listagem) em alto nível de filas de serviço prioritário no *Open vSwitch*. Formatos das URIs REST não foram apresentados	Topologia SDN virtual composta de 3 nós e 4 *switches Open vSwitch* operando com o *OpenFlow* 1.0	*Floodlight*	Não
20	[Zhang *et al.* 2015]	***Ryuo*** é um *framework* voltado para o controlador *Ryu* contendo uma interface cujas funções permitem otimizar o processo de tradução das instruções de controle repassadas à rede SDN	O corpo de funções ***Ryuo*** não foi apresentado; autores citaram apenas que o módulo *Local Service* desenvolvido é o responsável por reduzir o número de instruções de controle *OpenFlow* necessárias para orquestrar a rede	Topologia SDN composta de 2 nós e 3 equipamentos Pica8 P-3295	*Ryu*	Não
21	[Sieber *et al.* 2015]	***NSAL*** é uma camada de abstração que oferece ao gerente abstrações para gerenciamento de QoS em alto nível em redes SDN controladas pelo *OpenDaylight*	O corpo de funções não foi suficientemente detalhado. Autores citaram apenas que funções da ***NSAL*** foram capazes de programar a rede SDN para atender demandas de chamadas de aplicações VOIP	Avaliação realizada com a plataforma *StableNet* e topologias contendo dispositivos NEC PF5240, *Cisco Catalyst* 3650 e dispositivos *Open vSwitch*	*OpenDaylight*	Não
22	[Yan *et al.* 2015]	***HiQoS*** é uma aplicação que permite alocar fluxos de tráfego multimídia às filas de prioridades, efetuar o controle de reserva de banda e computar rotas de comunicação em redes SDN controladas pelo *Floodlight*	As funções usadas pelo gerente para envio de fluxos de pacotes às filas de prioridades e controle de reserva de banda não foram apresentadas; apenas o algoritmo desenvolvido pelos autores para computação de rotas é discutido. Não há integração entre ***HiQoS*** e a interface nativa do *Floodlight*: ***HiQoS*** instaura regras de controle diretamente nos dispositivos da rede SDN usando o protocolo OVSDB	Topologia SDN virtual instanciada no *Mininet* composta de 5 *switches*, 2 servidores e 11 nós virtuais	*Floodlight*	Não

Continua na próxima página

Tabela 3.1 – continuação da página anterior

	Trabalho	Descrição da solução	Detalhamento da interface	Ambiente de avaliação	Controlador(es)	Interoperabilidade
23	[Xu, Chen e Qian 2015]	Um *framework* voltado para o controlador *Ryu* tendo em vista o provisionamento simplificado de QoS em redes SDN	A interface do *framework* é capaz de: (a) configurar de filas de serviços em portas de dispositivos *OpenFlow*; e (b) alocar fluxos de tráfego prioritário às filas existentes, em tempo de execução. Detalhes quanto aos protótipos das funções não foram apresentados pelos autores	Topologias de redes SDN reais e ainda virtuais instanciadas no *Mininet* com dispositivos *Open vSwitch*	*Ryu*	Não
24	[Rivera, Fei e Griffioen 2016]	***RAPTOR*** consiste de uma interface *northbound* baseada em princípios REST capaz de suprimir detalhes internos de operação de dois controladores para programar a rede SDN	A interface é um protótipo inicial com apenas 3 funções (URIs REST): `(GET/(static)flows`, `DELETE/flows` e `GET/switches`). ***RAPTOR*** não possui uma função capaz de *instalar* uma regra de controle na rede: este processo depende da utilização de um módulo externo—*Flow Handler Module*—da plataforma GENI, que é voltada para instanciação dinâmica de ambientes de testes (*testbeds*) e experimentação em SDN virtuais	Simulação em topologia SDN virtual usando a plataforma GENI	*Floodlight* e *Ryu*	Sim, mas é preciso utilizar a plataforma GENI para instalar instruções de controle em um ambiente SDN inteiramente virtual
25	[Belzarena *et al.* 2016]	Uma arquitetura SDN de referência, com uma interface *northbound* voltada para aplicações que precisam programar a rede SDN tendo em vista atender determinadas demandas de QoS	O papel da interface é destacado na pesquisa. Contudo, os protótipos das funções e os parâmetros de controle da rede definidos no estudo não são suficientemente detalhados	Não informado	Não informado	Não
26	[Layeghy, Pakzad e Portmann 2016]	***SCOR*** é uma interface *northbound* voltada para o controlador POX capaz de acomodar requisitos de QoS na rede SDN	A a solução proposta consiste de uma linguagem declarativa e um conjunto de 8 preditores de alto nível para acomodar requisitos de QoS na rede. ***SCOR*** exige que o gerente descreva manualmente a estrutura do problema de QoS usando tais preditores, que serão compilados e repassados ao controlador POX	Experimentos realizados em topologias SDN virtuais	POX	Não

Continua na próxima página

Tabela 3.1 – continuação da página anterior

	Trabalho	Descrição da solução	Detalhamento da interface	Ambiente de avaliação	Controlador(es)	Interoperabilidade
27	[Pham e Hoang 2016]	Interface *northbound* baseada na intenção (*intent-based*) para redes SDN controladas pelo ONOS	A interface desenvolvida é um protótipo inicial e visa descrever os regras de controle usando estruturas próximas da linguagem natural. São destacados princípios que orientam a concepção de interfaces *intent-based*, porém, a arquitetura da API proposta não é detalhada. Afirma-se que haverá suporte aos formatos CLI e REST, mas os comandos e formatos das URIs não são suficientemente discutidos	Topologia SDN virtual instanciada no ambiente *Mininet*	ONOS	Não
28	[Sieber *et al.* 2016]	Protótipo inicial de uma camada de gerenciamento tendo em vista o controle padronizado de dispositivos *OpenFlow* e legados com abstrações de alto nível	A solução permite monitorar equipamentos de rede heterogêneos, além de verificar e acomodar requisitos de QoS em ambientes de rede SDN. Autores citam que a interface proposta baseia-se em princípios REST, mas sem apresentar os formatos das URIs definidas	Experimentos usam o *Panopticon* e 2 topologias de redes	Não informado	Não
29	[Chen e Wu 2017]	***SemSDN*** consiste de uma interface *northbound* inspirada em princípios de *Web* semântica para gerenciar recursos de redes SDN controladas pelo *Floodlight*	A solução usa a linguagem SPARQL e define preditores para programar a rede. As funções de *SemSDN* são acessadas por meio de *Servlets Java*. São 05 funções que permitem: (a) identificar *loops* na rede, (b) efetuar a computação de rotas, (c) identificar a topologia e recolher informações de QoS existentes na rede, (d) coletar funcionalidades de QoS disponíveis no equipamento *OpenFlow*, e (e) imprimir gráfico da topologia de rede	Topologia SDN virtual com 7 dispositivos *Open vSwitch* e outros 8 nós	*Floodlight*	Não
30	[Layeghy, Pakzad e Portmann 2017]	***SCOR*** é uma interface *northbound* voltada para o controlador POX capaz de acomodar requisitos de QoS na rede SDN	Continuação do trabalho [Layeghy, Pakzad e Portmann 2016]. ***SCOR*** oferece 2 novos preditores para controlar a SDN. Gerente ainda deve selecionar os preditores e construir (manualmente) todas as regras necessárias à programação da rede	Topologia SDN virtual com 7 nós	POX	Não

Continua na próxima página

Tabela 3.1 – continuação da página anterior

	Trabalho	Descrição da solução	Detalhamento da interface	Ambiente de avaliação	Controlador(es)	Interoperabilidade
31	[Pan *et al.* 2017]	***OpenShed*** consiste de uma arquitetura SDN com uma interface *northbound* para provisionamento de QoS em redes controladas pelo ONOS	A interface apóia-se no padrão de projeto *Builder* e suas funções são invocadas de modo semelhante a um construtor na linguagem C++. Um componente (*QoS Policy Parser* desenvolvido traduz as requisições da interface em comandos de baixo nível. São 03 funções que permitem: (a) realizar uma conexão e registrar um dispositivo de rede, (b) consultar o estado e colher informações quanto a porta do equipamento, e (c) obter a política de QoS do componente (*QoS Policy Parser*) e acomodá-la na SDN	Topologia SDN virtual com dispositivos *Open vSwitch*	ONOS	Não
32	[Neto *et al.* 2017]	***MISSIn*** consiste de uma ferramenta de apoio para a orquestração de infraestruturas SDN capaz de interagir com diferentes controladores	Apenas o papel da interface do MISSIn é destacado. Nenhum protótipo de função, parâmetros de controle disponíveis ao gerente ou modelos de dados usados na compatibilização com as interfaces nativas dos controladores são apresentados.	Nenhum tipo de experimento foi conduzido	*Floodlight* (suporte em fase experimental)	Não

3.2 Considerações Finais

Neste capítulo, foram apresentadas as principais abordagens que possuem relação com a interface *northbound* proposta neste trabalho.

Observando-se o quadro comparativo (Tabela 3.1), verifica-se que nenhum dos trabalhos encontrados na literatura permite a pronta reutilização da abordagem proposta em mais de um controlador. Em um único trabalho [Rivera, Fei e Griffioen 2016], a interoperabilidade da interface *northbound* depende da utilização e recursos de uma plataforma externa voltada para experimentação em ambientes SDN puramente virtuais. Ainda, poucos trabalhos mostram detalhes quanto aos protótipos das funções desenvolvidas, à integração com as interfaces nativas do controlador-alvo ou à forma de utilização das funções para programar a rede.

A maioria dos trabalhos encontrados na literatura, voltados para a especificação de interfaces *northbound* para redes SDN, está focada na seleção de parâmetros de controle variados, mas sem endereçar questões de reuso e interoperabilidade das funcionalidades entre controladores distintos. Em particular, para modificar o comportamento da rede SDN usando um outro controlador, gerentes são forçados a lidar com detalhes relacionados à utilização do novo corpo de funções existente, com o aprendizado de modelos de dados complexos, com as configurações e funcionamento do controlador. Os trabalhos apresentados neste capítulo confirmam esta condição.

Apesar de características de programação da rede com interfaces *northbound* já terem sido alcançadas pelas abordagens discutidas neste capítulo, a principal delas ainda se mantém em aberto: a interoperabilidade. Nenhuma destas abordagens foi concebida visando fornecer aos usuários-finais, gerentes e desenvolvedores de aplicações para redes SDN uma interface *northbound* com suporte à comunicação em alto nível com mais de um controlador e que priorize a acomodação simplificada e automatizada de requisitos de controle na rede, em tempo de execução, a partir de um corpo de funcionalidades uniforme e que possa ser reutilizado sem esforço.

Capítulo 4

A Interface *Northbound* NoBI

Neste capítulo, é apresentada uma abordagem alternativa para a programação dinâmica de redes definidas por *software*, chamada NoBI, com foco na configuração da rede em alto nível, bem como na comunicação transparente e garantia de interoperabilidade do corpo de funções da interface entre controladores SDN distintos.

4.1 Introdução

Conforme visto no Capítulo 3, pesquisas têm endereçado várias abordagens capazes de implantar regras de controle para reconfigurar o comportamento da rede SDN, usando interfaces *northbound*. Certamente, tais interfaces propostas são úteis, uma vez que permitem abstrair a elevada complexidade da infraestrutura física da rede. Cumpre ressaltar, porém, que nesta busca pela simplificação da instalação de regras de controle na rede, o projeto de interfaces de programação *northbound* com suporte à comunicação efetiva e transparente com controladores SDN distintos tem sido deixado em segundo plano.

Neste sentido, o presente capítulo propõe a utilização de interfaces *northbound* focadas na flexibilidade e interoperabilidade para tal finalidade. Na Seção 4.2, é levada a efeito uma discussão dos principais atributos que foram observados na implementação da interface proposta para programar redes SDN. A seguir, na Seção 4.3, é apresentada uma abordagem alternativa, inspirada em princípios arquiteturais REST [Fielding 2000], desenvolvida para simplificar a programação de redes SDN por parte dos seus utilizadores (e.g., gerentes), bem como assegurar a interoperabilidade do corpo de funções entre controladores de classe

empresarial distintos. Discute-se, ainda nesta seção, o projeto arquitetural da NoBI e um cenário de instanciação desta arquitetura. Em seguida, os modelos de dados agnósticos aos controladores usados no processo de representação de parâmetros de controle da rede também são discutidos. Ainda, são apresentadas determinadas particularidades relacionadas ao conjunto de serviços NoBI, com a descrição de cada uma das funções disponíveis ao gerente para programar a rede SDN em alto nível, além dos principais parâmetros de controle estabelecidos. Por fim, são apresentadas as considerações finais deste capítulo, na Seção 4.4.

4.2 Atributos da Abordagem NoBI

A NoBI satisfaz um conjunto de valores e requisitos considerados pelo autor da presente tese como essenciais para o desenvolvimento de interfaces *northbound*. Dentre eles, destacam-se:

- **Independência de controlador:** as funções de controle (Subseção 4.3.4) da interface NoBI são capazes de abstrair os detalhes internos de operação de controladores SDN distintos tendo em vista assegurar a interoperabilidade e maior flexibilidade do corpo de funcionalidades proposto. Com a NoBI, portanto, da mesma forma que o gerente pode definir qual controlador utilizar no momento de projetar a rede, torna-se mais simples substituí-lo, se necessário. Caso haja um cenário de evolução em que o controlador original necessite de ser trocado por outro, o impacto desta mudança sobre o gerente será reduzido, posto que a mesma interface continuará a ser usada para programar a rede. Assim, o esforço do gerente para lidar com a incompatibilidade e as diferenças entre as interfaces dos controladores é minimizado e transferido para a interface de controle NoBI. Além disso, as aplicações de *software* para ambientes SDN também podem se beneficiar desta independência de controlador. Isto porque estes sistemas têm sido tradicionalmente construídos considerando uma implementação *northbound* particular de controlador. Isto dificulta a inovação, além de trazer problemas para os desenvolvedores quando o plano de controle SDN precisar evoluir. Diferentemente de vários outros estudos publicados e discutidos no Capítulo 3, a pronta reutilização das funções da interface em controladores distintos, já contemplada pela abordagem NoBI, reforça ainda mais sua contribuição científica.

- **Programação dinâmica:** a abordagem NoBI buscou endereçar os requisitos de programabilidade e configurabilidade da SDN em tempo real. Particulamente, gerentes são capazes de interagir com o controlador existente em alto nível e instalar regras de controle na SDN em tempo de execução. Tal processo é automático e, ao contrário de outras soluções encontradas na literatura, não é necessário escrever códigos específicos de controladores ou aplicar instruções de forma manual via terminal de linha de comando para programar a rede.

- **Uso de modelos de dados unificados:** neste trabalho, modelos de dados foram desenvolvidos (Subseção 4.3.3) para representar os parâmetros de controle da rede disponíveis ao gerente. Especificamente, foram definidos modelos de dados para a identificação da regra a ser removida com a função (URI) `/nobi/controller/delete-flow` e para representação da regra para reserva de largura de banda usada da função (URI) `/nobi/controller/create-qos-path`. Estes modelos de dados, juntamente com as demais URIs NoBI (Subseção 4.3.4), garantem a plena interoperabilidade das operações realizadas, permitindo ao gerente trabalhar com um conjunto padronizado de *tags*, em vez de lidar com detalhes complexos de representação de instruções dos controladores. Cada uma das *engines* propostas no projeto arquitetural (Subseção 4.3.1) NoBI é que tem o papel de traduzir os parâmetros de controle da rede definidos pelo gerente em instruções específicas do controlador-alvo existente.

- **Projeto baseado em REST:** alguns princípios REST (do inglês *Representational State Transfer*) [Fielding 2000] também foram incorporados à solução proposta, na intenção de utilizar boas práticas deste estilo arquitetural que tem se consolidado fortemente no desenvolvimento de diversas APIs públicas, e.g., [Google Inc. 2016; Facebook Inc. 2016]. Particularmente, a NoBI se destaca de outras interfaces *northbound* oferecidas por alguns controladores empresariais conhecidos, as quais exibem os tipos de dados (diretamente) nas URIs de acesso a recursos ou usam métodos HTTP de modo inconsistente e, portanto, violam alguns dos preceitos fundamentais do REST. Por exemplo, na API do *Floodlight* [Izard 2017], o princípio REST que estabelece que o formato (ou tipo) dos recursos deve ser escolhido por meio da negociação de conteúdo entre o serviço e o cliente não é respeitado. Uma URI REST, portanto, não deve estar

permanentemente associada (acoplada) a uma representação particular do recurso. Já na API do controlador *Ryu* [Ryu Development Team 2017], outro princípio REST não observado é o que diz que as operações (verbos) devem ser orientadas ao protocolo utilizado. O *Ryu* usa o método `POST` do protocolo HTTP na URI */stats/flowentry/delete* para remover regras existentes, quando o mais coerente seria utilizar o método HTTP `DELETE`. A NoBI, por outro lado, buscou respeitar tais princípios REST em sua implementação.

4.3 A Interface *Northbound* NoBI

4.3.1 Arquitetura

Na Figura 4.1, ilustra-se o projeto da arquitetura da abordagem NoBI, a partir de um diagrama de classes simplificado, que destaca as principais entidades funcionais que formam o núcleo do arcabouço da NoBI e seus relacionamentos.

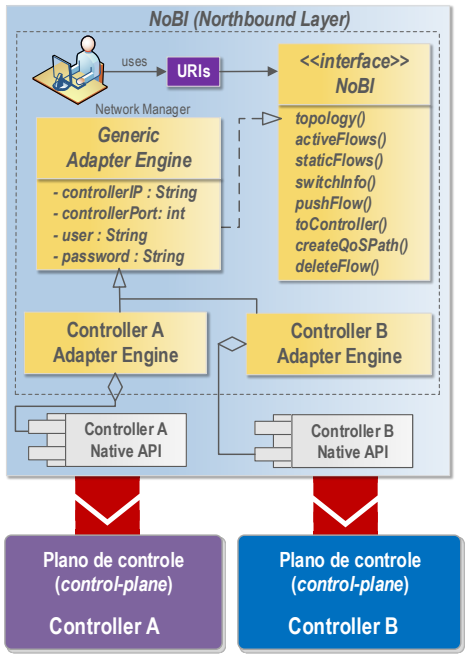


Figura 4.1: Arquitetura da interface *northbound* NoBI em UML

Primeiramente, observe que a base do projeto arquitetural é o padrão de projeto *Adapter* [Gamma *et al.* 1995], que permite converter a interface de um recurso existente à outra

interface-alvo requerida pelo cliente. Neste trabalho, a interface-alvo é representada pela entidade `NoBI` da Figura 4.1. As seguintes entidades e papéis são definidos no projeto NoBI:

- `NoBI`: entidade responsável por especificar um contrato de interface, ou seja, um conjunto de funcionalidades que devem ser respeitadas e implementadas por outras entidades. Este conjunto de funções é acessado pelos gerentes de forma indireta via requisições HTTP/REST às URIs (vide Seção 4.3.4) para consultar informações e instalar/desinstalar regras de controle na rede SDN;

- `Generic Adapter Engine`: apesar de não poder ser instanciada e executada, esta entidade é responsável por fatorar código comum às *engines* adaptadoras existentes. Dentre os principais serviços disponíveis às subclasses, pode-se destacar métodos para manipulação dos endereços, autenticação e portas de acesso dos controladores, além de serviços disponíveis para leitura/escrita de arquivos nos formatos XML e JSON;

- `Controller Adapter Engine`: responsável por traduzir, em tempo de execução, as requisições feitas às funções (URIs) NoBI em instruções específicas do controlador. Além de implementar o contrato definido pela entidade `NoBI`, esta *engine* deve colocar-se entre as interfaces nativas do controlador-alvo e o gerente, adaptando-as e escondendo toda a complexidade, variabilidade e como as abstrações de controle são implementadas pelo controlador;

- `Controller Native API`: esta entidade representa a API *northbound* original do controlador-alvo que será adaptada. Seu papel é disponibilizar à *engine* adaptadora correspondente as funcionalidades e abstrações nativas para consulta/instalação/remoção de instruções na rede SDN.

4.3.2 Cenário de Instanciação

Uma vez estabelecida a arquitetura e definidas as entidades funcionais e seus papéis, pode-se descrever um cenário de execução detalhado com a instanciação da abordagem NoBI.

Com base na delimitação do escopo do suporte ferramental usado neste trabalho em relação aos controladores SDN, já discutida na Subseção 2.3.1 do Capítulo 2, definiu-se um cenário de execução com interface NoBI proposta, o qual é ilustrado na Figura 4.2. Especificamente, a

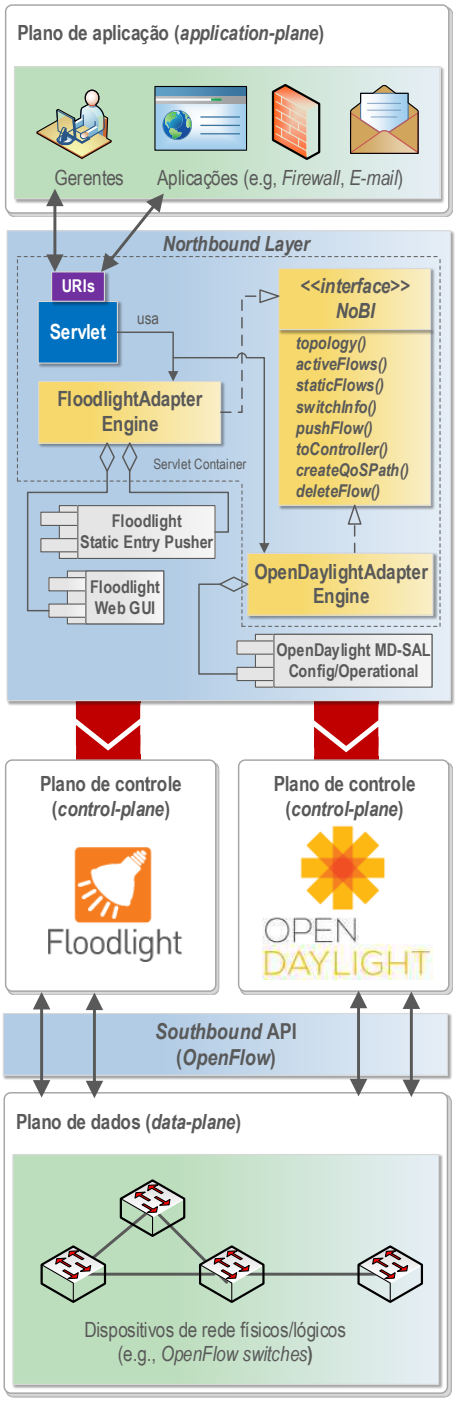


Figura 4.2: Composição de um cenário de execução SDN com a instanciação da abordagem *northbound* NoBI proposta

construção deste cenário foi usada como prova de conceito com a implantação da NoBI sobre um ambiente de rede SDN real usando alguns controladores de classe empresarial amplamente utilizados: *Floodlight* [Big Switch Networks 2013] e *OpenDaylight* [OpenDaylight 2013].

Além destes controladores, diferentes equipamentos *OpenFlow* foram utilizados em uma proposta particular de avaliação da abordagem NoBI, para observar seu comportamento e demonstrar sua viabilidade (como será apresentada no Capítulo 5).

Primeiramente, na Figura 4.2, note que os componentes *Floodlight Static Entry Pusher* [Izard 2017] e *Floodlight Web GUI* [Izard 2017a] são nativos do *Floodlight*. Basicamente, estes oferecem aos gerentes funcionalidades para instalar instruções na rede e visualizar informações da SDN em tempo real em uma interface *Web*. Cumpre ressaltar, porém, que, apesar de úteis, as interfaces disponíveis para programar a rede são complexas e devem ser adaptadas, pois, além de não permitirem a composição de regras de controle reutilizáveis, transferem para seus usuários o ônus de implementar (manualmente) as instruções por meio de estruturas de baixo nível e que, em alguns casos, dependem de versão *OpenFlow* específica.

No âmbito do controlador *OpenDaylight*, a infraestrutura *Model-Driven Service Abstraction Layer* (MD-SAL) [OpenDaylight 2017b] tem o papel de disponibilizar ao gerente as APIs *northbound*, gerenciar as estruturas de dados utilizadas por vários serviços do controlador e oferecer acesso a recursos especializados. Não obstante a existência de outros subsistemas no MD-SAL, dois deles, em particular, destacam-se: *Config* e *Operational*. O primeiro é capaz de receber e armazenar todas as configurações e instruções instaladas pelo gerente por meio da API *northbound* do controlador, enquanto que o segundo tem o papel de apresentar as informações quanto ao estado atual do *OpenDaylight* e da infraestrutura da SDN. Tais informações são normalmente coletadas pelo gerente e não podem ser modificadas via API *Operational*. Similarmente ao *Floodlight*, as abstrações de controle do *OpenDaylight* são basicamente orientadas à implementação e não-reutilizáveis, isto é, exigem que o gerente elabore as regras de controle da SDN de forma manual e a partir de instruções específicas do controlador. Além disso, apesar do foco em alguns princípios REST, as APIs *northbound* do *OpenDaylight* não são capazes de comunicar-se de maneira transparente com outro controlador SDN distinto.

Assim sendo, buscando definir uma interface *northbound* alternativa mais flexível e com suporte à interoperabilidade, foi levada a efeito a construção de duas *engines* compatíveis com as interfaces oferecidas pelos controladores *Floodlight* e *OpenDaylight*. No caso do *Floodlight*, em particular, a *engine* correspondente é representada na Figura 4.2 pela classe `FloodlightAdapterEngine`, enquanto que a `OpenDaylightAdapterEngine` é capaz de comunicar-se com o controlador *OpenDaylight*. Conforme citado anteriormente, além de

implementar o mesmo contrato definido pela interface-alvo `NoBI`, o papel fundamental de cada *engine* é colocar-se entre os subsistemas do controlador correspondente e o plano de aplicação SDN, adaptando as interfaces (de baixo nível) e também escondendo do gerente toda a complexidade, detalhes operacionais e como as abstrações de controle são implementadas em cada controlador (*Floodlight*/*OpenDaylight*, nesta instanciação particular).

O conceito de adaptador descrito em [Gamma *et al.* 1995] estabelece que, além de implementar a interface-alvo, o adaptador deve ser capaz referenciar o componente que terá sua interface adaptada. Isto porque, ao receber uma solicitação de execução de um serviço, o adaptador irá delegar para o componente referenciado a execução deste serviço utilizando a interface original. No caso do controlador *Floodlight*, o processo de inserção do adaptador `FloodlightAdapterEngine` sobre os componentes *Floodlight Web GUI* e *Floodlight Static Entry Pusher* é mostrado na Figura 4.2. Observe que a *engine* adaptadora implementada pode ser vista como um contêiner que possui dois componentes funcionais como entidades filhas. Dessa forma, a execução do serviço solicitado é levada a efeito por uma implementação original do subsistema do *Floodlight*, mas que está sendo adaptada pela classe `FloodlightAdapterEngine`. O mesmo conceito se aplica à *engine* adaptadora `OpenDaylightAdapterEngine`, porém, os subsistemas-alvo referenciados e invocados para execução de serviços são MD-SAL *Config* e MD-SAL *Operational*.

Uma vez que buscou-se incluir alguns princípios arquiteturais REST [Fielding 2000] na abordagem NoBI proposta, fez-se necessário permitir o acesso às funções de controle da interface-alvo por meio de URIs REST. A ideia, portanto, é oferecer ao gerente um conjunto de URLs padronizadas, compatíveis com os dois controladores SDN selecionados e que assegurem a interoperabilidade das abstrações da interface NoBI. Para tanto, as entidades `NoBI`, `OpenDaylightAdapterEngine` e `FloodlightAdapterEngine` foram incorporadas a uma aplicação *Web* escrita na linguagem *Java*. Dessa forma, a partir do que é mostrado na Subseção 4.3.4 e no detalhamento realizado no Apêndice A, o leitor perceberá que, do ponto de vista de utilização, o gerente deve apenas acessar funções (URIs) da interface NoBI proposta e enviar mensagens HTTP, para orquestrar o controlador SDN existente e programar a rede. Particularmente, pode-se dizer que este conjunto de URIs (Subseção 4.3.4) substancia a abordagem NoBI em si, pois provê um ponto de acesso para o gerente com abstrações independentes do controlador. Especificamente, tais URIs abstraem os detalhes de funcionamento

internos do *Floodlight* e *OpenDaylight* e permitem reconfigurar o comportamento da rede SDN em alto nível.

Na Figura 4.2, ainda, note que um programa no lado servidor (*Servlet*[1]) é utilizado no mapeamento da URI solicitada a uma implementação concreta da interface-alvo `NoBI`. Sendo assim, o núcleo do projeto arquitetural NoBI pode ser mantido simples, pois: i) a existência dos adaptadores é "escondida" do gerente de redes; ii) a *engine* adaptadora atual pode ser trocada por outra que implemente a interface-alvo, em tempo de execução e sem que haja impacto inoportuno no usuário ou aplicação-cliente; e iii) caso haja mudanças no subsistema do controlador, estas serão encapsuladas apenas na classe adaptadora, e não percebidas por aqueles que acessam a interface-alvo.

Quanto à flexibilidade e extensibilidade do núcleo do projeto de classes, é importante frisar que, na presente instanciação, o corpo de funções da abordagem NoBI oferece suporte apenas àquelas redes SDN controladas pelo *Floodlight* e *OpenDaylight*. Entretanto, o *design* de classes da Figura 4.1 pode evoluir e ser facilmente modificado para acomodar novas classes adaptadoras e assim oferecer suporte a outros controladores. Isso porque a base do projeto de classes permite que as URIs de controle estejam desacopladas das interfaces específicas de cada controlador. Em particular, o único requisito fundamental é que o contrato de métodos definido pela interface-alvo `NoBI` seja implementado por uma classe adaptadora, quando se deseja oferecer suporte a um novo controlador SDN.

4.3.3 Modelos de Dados

Nesta seção, é descrito o processo de criação do modelo de dados genérico NoBI. Em particular, este modelo contém as estruturas de comandos agnósticas ao controlador existente as quais são usadas pelo gerente para repassar parâmetros de entrada para programar a rede.

Os controladores SDN *Floodlight* e *OpenDaylight* não apenas definem seus próprios formatos de arquivo padrão (JSON ou XML, por exemplo) e abstrações de controle, mas, também, suas próprias estruturas para representar dados (e.g., nomenclatura de *tags*, campos e instruções). Essa autonomia torna-se, ao mesmo tempo, a principal barreira para criação de interfaces *northbound* interoperáveis; isto porque as estruturas de informação recebidas ou

[1] O modelo de requisição e resposta HTTP com *Servlet* está disponível nos pacotes Java *javax.servlet* e *javax.servlet.http*.

retornadas por um controlador são, normalmente, completamente diferentes de outro, trazendo incompatibilidade e dificultando a comunicação das funções da interface entre diferentes controladores. Portanto, para assegurar a interoperabilidade da NoBI entre os controladores, definiu-se para a primeira função **`/nobi/controller/create-qos-path`** um conjunto de estruturas (modelo de dados) unificado e apropriado para o processo de representação dos parâmetros de entrada. Este modelo foi compatibilizado com os controladores suportados pela NoBI e é apresentado na Tabela 4.1.

Tabela 4.1: Consolidação dos modelos de dados nativos dos controladores para um modelo unificado suportado pela NoBI para a URI `/nobi/controller/create-qos-path`

Descrição das Instruções	**Modelo de Dados Nativo *OpenDaylight***	**Modelo Dados Nativo *Floodlight***	**Modelo de Dados Unificado NoBI**
URI de acesso	/restconf/config/opendaylight-inventory:nodes/node/	/wm/staticflowpusher/json	/nobi/controller/create-qos-path
Método HTTP padrão	PUT	POST	POST
Formato padrão do arquivo de comandos	XML	JSON	JSON
Sintaxe do comando usado para identificar a instrução	<id></id>	"name":" "	"id":" "
Sintaxe dos comandos usados para identificar o tipo do protocolo	<match> <ethernet-match> <ethernet-type> <type></type> </ethernet-type> </ethernet-match> <ip-match> <ip-protocol></ip-protocol> </ip-match> </match>	"eth_type": " " "ip_proto": " "	"eth_type": " " "ip_proto": " "
Sintaxe dos comandos usados para identificar os *hosts* de origem e destino	<match> <ethernet-match> <ethernet-source> <address></address> </ethernet-source> <ethernet-destination> <address></address> </ethernet-destination> </ethernet-match> <ipv4-source></ipv4-source> <ipv4-destination></ipv4-destination> </match>	"eth_src": "" "eth_dst": "" "ipv4_src": "" "ipv4_dst": ""	"ipv4_src": " " "ipv4_dst": " "
Sintaxe do comando usado para definir a classe de serviço do tráfego prioritário marcado	<ip-dscp></ip-dscp>	“ip-dscp”: “”	“ip-dscp”: “”
Sintaxe do comando para configurar uma reserva (mínima) de largura de banda em uma fila de serviço (kbps)	<set-queue-action> <queue></queue> <queue-id></queue-id> </set-queue-action>	“actions”: “set_queue=”	“min_bandwidth_kbps” : “”

Similarmente à URI **`/nobi/controller/create-qos-path`** anterior, foi necessário definir uma forma padronizada para representar o parâmetro de entrada requerido pela função **`/nobi/controller/delete-flow`**, que consiste de um identificador único da regra que se deseja remover, isto é, um `<id>`. O cliente desta função NoBI, para tanto, deve fornecer como argumento este `<id>` da regra a ser removida no formato JSON, quando efetuar a requisição HTTP do tipo DELETE. Este modelo de dados criado é mostrado na tabela Tabela 4.2.

Tabela 4.2: Consolidação dos modelos de dados nativos dos controladores para um modelo unificado suportado pela NoBI para a URI `/nobi/controller/delete-flow`

Descrição das Instruções	**Modelo de Dados Nativo *OpenDaylight***	**Modelo Dados Nativo *Floodlight***	**Modelo de Dados Unificado NoBI**
URI de acesso	/restconf/config/opendaylight-inventory:nodes/node/<switch-dpid>table/<tbl-num>/flow/<identificador>	/wm/staticflowpusher/json	/nobi/controller/delete-flow
Método HTTP padrão	DELETE	DELETE	DELETE
Formato padrão do arquivo de comandos	-	JSON	JSON
Sintaxe do comando usado para identificar a instrução a ser removida	*(identificadores do equipamento, tabela e regra são passados manualmente na URI)*	"name":" "	"id":" "

O resultado deste processo de consolidação foi um conjunto de estruturas para representar dados padronizado e que pode ser usado pelo gerente de maneira intercambiável com o *Floodlight* e *OpenDaylight*. Em particular, as *engines* implementadas neste trabalho traduzem as estruturas do modelo de dados NoBI unificado em instruções específicas dos modelos suportados pelo controlador existente, de forma automática. As estruturas NoBI definidas podem ser estendidas sem grandes esforços. Para tanto, basta que sejam traduzidas em instruções nativas por uma nova classe adaptadora correspondente, quando se deseja oferecer suporte a um novo controlador SDN.

4.3.4 Definição do Arcabouço de Funcionalidades

Nesta seção, é levada a efeito uma descrição das principais funcionalidades da abordagem proposta neste trabalho. Entretanto, uma discussão mais detalhada sobre a forma de utilização de cada uma das funções oferecidas pela interface NoBI pode ser encontrada no Apêndice A.

Conforme mencionado anteriormente, além de endereçar o problema da ausência de uma camada de abstração para programação de redes SDN independente de controlador, outro objetivo importante da solução proposta neste trabalho é prover ao gerente um conjunto de

abstrações (URIs) que permita programar a SDN de forma simples e capaz de comunicar-se de forma transparente com diferentes controladores. Da perspectiva de utilização, tais URIs representam o corpo de funções da interface NoBI e, apesar de não oferecerem suporte a todos os controladores SDN correntes, estas foram compatibilizadas com alguns representantes existentes de controladores de classe empresarial de código aberto: o *Floodlight* v1.2 [Big Switch Networks 2013] e *OpenDaylight Beryllium-SR4* [OpenDaylight 2013]. As abstrações de controle NoBI retiram do gerente de redes a exaustiva e frequente tarefa de implementar código capaz de lidar com a complexidade e as diferenças entre as APIs dos controladores para assegurar a capacidade de portar (reutilizar) as funções de programação da rede.

Descrição das Funções da Interface *Northbound* NoBI

A seguir, é feita uma descrição geral de cada uma das funções (URIs) disponíveis ao gerente para programar a rede SDN:

- **`/nobi/controller/topology`:** a presente função permite ao gerente de redes comunicar-se com o controlador SDN existente (*Floodlight* ou *OpenDaylight*) e requisitar detalhes (*hosts*, *switches*, portas, etc.) quanto à topologia SDN. Posto que esta função não exige a passagem de parâmetros, uma requisição HTTP simples com o método GET deve ser feita da seguinte forma:

```
GET /nobi/controller/topology HTTP/1.1
Accept: application/json
```

  O leitor poderá notar que esta função não expõe o formato de arquivo (e.g., JSON ou XML) diretamente na URI. Essa característica permite que o recurso identificado por uma URI (a topologia, neste caso) possa ser apresentado em outro formato. A escolha do formato desejado do recurso é feita por meio da negociação de conteúdo entre o cliente da URI e o serviço disponível na NoBI, via uma indicação do formato (tipo) do arquivo no cabeçalho HTTP, para permitir que cliente e a interface NoBI tenham liberdade para evoluir, independentemente. Por padrão, a função irá retornar um código no formato JSON [Bray 2014]. A requisição GET anterior retornará uma resposta HTTP com o código 200 e um arquivo JSON contendo as informações da topologia SDN (vide Apêndice A, para maiores detalhes).

- **`/nobi/controller/active-flows:`** esta função permite ao gerente de redes interagir com o controlador SDN existente (*Floodlight* ou *OpenDaylight*) tendo em vista obter uma listagem detalhada de todas as regras de controle instaladas e ativas na rede. Posto que esta função não exige a passagem de argumentos, uma requisição HTTP simples com o método GET deve ser feita da seguinte forma:

```
GET /nobi/controller/active-flows HTTP/1.1
Accept: application/json
```

  Por padrão, a função irá retornar um código no formato JSON. A requisição anterior retornará uma resposta HTTP com o código 200 e um arquivo JSON contendo todas as regras de controle ativas na rede SDN (vide Apêndice A, para maiores detalhes).

- **`/nobi/controller/static-flows:`** esta função permite listar apenas as regras de controle instaladas pelo gerente na rede SDN, em vez daquelas instaladas (de forma automática) pelo controlador SDN. Não é exigida do cliente da função a passagem de parâmetros e a requisição HTTP deve ser levada a efeito da seguinte forma:

```
GET /nobi/controller/static-flows HTTP/1.1
Accept: application/json
```

  Por padrão, a função irá retornar um código no formato JSON. A requisição anterior retornará uma resposta HTTP com o código 200 e um arquivo JSON com todas as regras não instaladas de modo reativo pelo controlador na rede.

- **`/nobi/controller/switches:`** esta função tem por objetivo comunicar-se com o controlador SDN e requisitar informações acerca dos *switches* existentes no plano de dados SDN. Não é exigida do cliente da função a passagem de parâmetros e uma requisição HTTP simples pode ser levada a efeito da seguinte forma:

```
GET /nobi/controller/switches HTTP/1.1
Accept: application/json
```

  Por padrão, a função irá retornar um código no formato JSON (vide Apêndice A, para maiores detalhes).

- **`/nobi/controller/push-flow`:** esta é uma função de suporte que permite ao gerente instalar instruções na rede, garantindo-lhe apenas a liberdade necessária para elaborar regras de controle e acomodá-las na SDN quando estas *não* forem suportadas por funções nativas da interface NoBI. Isto evita que todas as instruções existentes do controlador precisem ser migradas para a NoBI, sobretudo em um instante inicial de sua implantação. Nesta função, contudo, deve-se escrever a regra utilizando as estruturas (*tags*) e o formato de arquivo padrão específicos suportados pelo controlador-alvo. O gerente deve fornecer como parâmetro o corpo da regra de controle e uma indicação do formato do arquivo no cabeçalho HTTP, quando efetuar a requisição HTTP do tipo POST. As regras de controle são repassadas ao *Floodlight* no formato JSON, enquanto que estas devem ser descritas no formato XML para o *OpenDaylight*. As requisições HTTP POST devem ser efetuadas da seguinte forma:

  Floodlight:

```
POST /nobi/controller/push-flow HTTP/1.1
Accept: application/json
Content-type: application/json
```

  OpenDaylight:

```
POST /nobi/controller/push-flow HTTP/1.1
Accept: application/xml
Content-type: application/xml
```

  Quando uma entidade é criada no servidor com o método POST, a especificação do protocolo HTTP [Fielding *et al.* 1999] informa que o código HTTP 201 ("*Created*") deve ser retornado com, opcionalmente, uma representação da entidade na resposta. Optou-se, pois, por retornar uma resposta HTTP contendo todo o código da regra recém-instalada no controlador (*Floodlight* ou *OpenDaylight*), juntamente com o código HTTP 201 (vide Apêndice A, para maiores detalhes).

- **`/nobi/controller/send-to-controller`:** a especificação do protocolo *OpenFlow* versão 1.3.0 [ONF 2012] afirma que, quando fluxos de pacotes não são tratados por nenhuma das regras de decisão existentes no equipamento de rede, estes serão descartados

(*dropped*), por padrão. A mesma especificação cita, porém, que este comportamento pode ser modificado criando-se uma regra para tratar estes casos[2]. Neste contexto, a abordagem NoBI oferece ao gerente uma função capaz de instalar uma regra para, por padrão, efetuar o encaminhamento dos fluxos de pacotes que não tiveram uma regra de decisão correspondente para o controlador. Esta função não exige do gerente a passagem de parâmetros. Apenas uma simples requisição HTTP do tipo POST deve ser feita da seguinte forma:

```
POST /nobi/controller/send-to-controller HTTP/1.1
Accept: application/json,application/xml
```

Quando uma entidade é criada no servidor com o método POST, a especificação do protocolo HTTP [Fielding *et al.* 1999] informa que o código HTTP 201 ("*Created*") deve ser retornado com, opcionalmente, uma representação da entidade na resposta. Optou-se, pois, por retornar uma resposta HTTP contendo todo o código da regra de encaminhamento recém-instalada no controlador (*Floodlight* ou *OpenDaylight*), juntamente com o código HTTP 201 (vide Apêndice A, para maiores detalhes).

- **`/nobi/controller/create-qos-path`:** considerando que diferentes classes de tráfego (e.g., voz ou vídeo) podem ser tratadas de formas diferentes na rede em vez de um único serviço de melhor esforço, esta função NoBI permite aplicar uma regra no controlador para programar a SDN e efetuar a reserva de largura de banda na comunicação entre dois pontos de origem/destino da rede. Especificamente, a partir da classificação de pacotes, o gerente poderá, em alto nível, solicitar o encaminhamento do tráfego marcado para uma fila de serviço e garantir uma taxa de vazão mínima. Por exemplo, pacotes pertencentes a uma aplicação multimídia poderiam ter prioridade sobre outros pacotes para receber um serviço diferenciado na rede.

  Neste cenário, esta função se mostra bastante útil no mapeamento do tráfego multimídia para uma fila de serviço e auxilia o gerente na configuração e estabelecimento de QoS em alto nível na SDN. A fila para a qual o tráfego será direcionado é obtida com base na classe de serviço do tráfego (um parâmetro para a função) e a configuração padrão de mapeamento (DSCP $\Longrightarrow$ Fila) do equipamento *OpenFlow* que receberá a solicitação.

[2] A mesma especificação *OpenFlow* fornece um nome especial a esta regra: *"table-miss flow entry"*.

A função utiliza um modelo de representação de dados desenvolvido, que é apresentado na Subseção 4.3.3. As estruturas NoBI para representação de dados dos parâmetros incluem: i) o endereço IP da origem `<ipv4_src>`; ii) o IP de destino `<ipv4_dst>`; iii) um identificador de regra `<id>` único, escolhido pelo gerente; iv) o código DSCP [Nichols *et al.* 1998] indicando a classe de serviço do tráfego que terá reserva de banda controlada `<ip_dscp>`; v) o tipo do protocolo `<eth_type>` [IANA 2017a]; vi) o identificador do protocolo da camada superior `<ip_proto>` [IANA 2017][3]; e vii) a reserva (mínima) de largura de banda desejada `<min_bandwidth_kbps>`, em kbps. O cliente da função deve fornecer estes parâmetros e utilizar o formato padrão JSON, quando efetuar a requisição HTTP do tipo POST. Por exemplo, para instalar no controlador (*Floodlight* ou *OpenDaylight*) uma regra identificada por `<id>=99` que visa a reserva de largura de banda mínima de 12 Mbps apenas para o tráfego UDP marcado com DSCP 34 (i.e., $AF41$) na comunicação entre dois *hosts* da rede com endereços de origem `<ipv4-src>=192.168.24.10` e destino `<ipv4-dst>=192.168.24.24`, deve-se fazer a seguinte requisição HTTP:

```
POST /nobi/controller/create-qos-path HTTP/1.1
Accept: application/json
Content-type: application/json
```

```
{
  "id":"99",
  "eth_type": "2048",
  "ip_proto": "17",
  "ipv4_src": "192.168.24.10",
  "ipv4_dst": "192.168.24.24",
  "ip_dscp": "34",
  "min_bandwidth_kbps": "12000"
}
```

Quando uma entidade é criada no servidor com o método POST, a especificação do protocolo HTTP [Fielding *et al.* 1999] informa que o código HTTP 201 ("*Created*") deve ser retornado com, opcionalmente, uma representação da entidade na resposta. Optou-se, pois, por retornar uma resposta HTTP contendo todo o código da regra recém-instalada no controlador (*Floodlight* ou *OpenDaylight*), juntamente com o código HTTP 201 (vide Apêndice A, para maiores detalhes).

[3] Por exemplo, um valor 6 indica que a porção de dados do datagrama IP será passada ao TCP, enquanto um valor 17 indica que os dados serão passados ao UDP.

- **`/nobi/controller/delete-flow`:** esta função permite remover uma regra de controle existente no controlador (*Floodlight* ou *OpenDaylight*), a partir de um identificador único da regra que se deseja remover. A presente função utiliza um modelo de representação de dados desenvolvido, que é apresentado na Subseção 4.3.3. A *engine* correspondente traduzirá este identificador de regra recebido em instruções de remoção específicas suportadas pelo controlador SDN existente, de forma automática.

  Por exemplo, para excluir do controlador a regra identificada por `ABC123`, o cliente deve efetuar a requisição HTTP contendo o seguinte código JSON como parâmetro:

```
DELETE /nobi/controller/delete-flow HTTP/1.1
Accept: application/json
Content-type: application/json
```

```
1  {"id":"ABC123"}
```

  Para a presente função, alinhado à especificação do protocolo HTTP [Fielding *et al.* 1999], buscou-se retornar o código HTTP 200 ("OK"), se a resposta incluir a entidade que representa o resultado (i.e., a regra recém-removida do controlador), ou o código 204 ("*No Content*"), em caso contrário.

Para efeito de sumarização, os protótipos das funções anteriormente descritas da interface *northbound* proposta, bem como os principais parâmetros definidos, quando aplicáveis, são mostrados na Tabela 4.3 adiante. Reitera-se que outros detalhes relacionados à forma de execução das funções, a passagem de parâmetros e exemplos de possíveis respostas emitidas por cada uma das funções são discutidos no Apêndice A, localizado no final deste documento.

Tabela 4.3: Listagem das funções da interface *northbound* NoBI proposta

URI NoBI	**Método HTTP**	**Parâmetros Requeridos**
`/nobi/controller/topology`	GET	–
`/nobi/controller/active-flows`	GET	–
`/nobi/controller/static-flows`	GET	–
`/nobi/controller/switches`	GET	–
`/nobi/controller/push-flow`	POST	O corpo da regra no formato JSON ou XML
`/nobi/controller/send-to-controller`	POST	–

`/nobi/controller/create-qos-path`	POST	`id`
		`eth_type`
		`ip_proto`
		`ipv4_src`
		`ipv4_dst`
		`ip_dscp`
		`min_bandwidth_kbps`

`/nobi/controller/delete-flow`	DELETE	`id`

4.4 Considerações Finais

Foi apresentada, neste capítulo, uma interface *northbound* para a programação de redes definidas por *software*, chamada NoBI, que prioriza uma maior flexibilidade, a partir da garantia de interoperabilidade de seu corpo de funcionalidades frente a planos de controle SDN distintos. A abordagem aqui proposta pode ser considerada como uma alternativa ao projeto convencional destas interfaces, realizado por entidades que mantém controladores SDN e trabalhos encontrados na literatura atualmente.

Da mesma forma que várias implementações de APIs (interfaces) disponíveis, a solução desenvolvida nesta tese contempla alguns dos objetivos fundamentais de uma interface *northbound* para redes SDN, como, por exemplo: ser capaz de abstrair a complexidade da infraestrutura de rede e permitir que o plano de aplicações SDN programe a rede em alto nível. Entretanto, diferentemente de vários outros estudos publicados, a NoBI é inovadora, posto que ela prevê mudanças no plano de controle SDN e assegura a interoperabilidade, isto é, a comunicação direta e transparente com mais de um controlador. Em particular,

as abstrações de controle NoBI são reutilizáveis e podem programar a rede em tempo de execução, mas sem que seus usuários tenham de arcar com o ônus de tratar a variabilidade (e complexidade) das funções de controle, instruções e modelos de dados nativos de cada controlador. Dessa forma, a dificuldade de operar controladores distintos e o impacto de uma mudança do plano de controle SDN sobre o utilizador são reduzidos, pois o esforço para lidar com a incompatibilidade entre as interfaces nativas dos controladores é desviado para a NoBI. A NoBI, no entanto, não oferece suporte a todos os controladores SDN modernos, mas, como prova de conceito, sua arquitetura foi instanciada e seu corpo de funcionalidades foi integrado a alguns dos melhores representantes existentes de *frameworks* de gerência SDN de classe empresarial e código aberto: o *Floodlight* versão 1.2 e *OpenDaylight Beryllium-SR4*.

Viu-se que o núcleo do projeto NoBI foi concebido tendo em vista a comunicação com os controladores em alto nível para instalar determinadas regras de controle na rede SDN via requisições HTTP/REST. Este processo é automático, não é necessário ter de escrever algum código específico do controlador ou ainda aplicá-lo em um terminal de linha de comando. Além da maior flexibilidade e interoperabilidade, outro diferencial da NoBI em relação às outras abordagens recentes para programar redes SDN inclui a definição de modelos de dados unificados para representação dos parâmetros de entrada de funções.

Ainda, posto que as operações de QoS em alto nível nas redes atuais são problemas que se mantém em aberto [Karakus e Durresi 2017] e o uso de filas de serviços é visto por alguns autores como um tópico não tratado adequadamente no paradigma SDN [Pan *et al.* 2017], buscou-se oferecer, no corpo de abstrações NoBI, uma função de controle voltada para o encaminhamento do tráfego prioritário para uma fila de serviço, visando auxiliar o gerente na configuração de QoS em alto nível na SDN. A programação da rede SDN com esta finalidade mostra-se uma atividade útil e vantajosa para aplicações que operam com a transmissão de tráfego multimídia entre pontos da rede (e.g., jogos com múltiplos jogadores).

Discutiu-se, inicialmente, importantes requisitos (atributos) incorporados à concepção da interface *northbound* proposta. Em seguida, não apenas foi apresentado o núcleo do projeto da abordagem, mas, as principais entidades funcionais foram detalhadas (descrição e papel desempenhado). Ainda, mostrou-se de forma detalhada um cenário de instanciação da arquitetura NoBI sobre um ambiente SDN real com controladores distintos, juntamente com os modelos de dados desenvolvidos. Por fim, foi levada a efeito uma descrição geral de cada

uma das principais funcionalidades oferecidas pela abordagem NoBI para operar a rede SDN.

O próximo capítulo apresenta ao leitor um estudo experimental conduzido para observar o comportamento da abordagem proposta. Em particular, é investigado o potencial de uso das funções inicialmente oferecidas pela interface NoBI para programar a rede e atender requisitos específicos de aplicações que necessitam operar com a transmissão de vídeo entre pontos remotos da rede, além de uma análise qualitativa com usuários reais.

Capítulo 5

Avaliação Experimental e Resultados

Para demonstrar a viabilidade da abordagem NoBI, este capítulo descreve um estudo experimental conduzido sobre topologias de redes SDN reais, contendo equipamentos e controladores de classe empresarial distintos, além de uma análise qualitativa.

Neste contexto, uma vez que o principal objetivo de pesquisa deste trabalho é a proposição uma interface *northbound* com suporte à comunicação transparente e garantia de *interoperabilidade* do seu corpo de funções entre controladores SDN, o experimento busca principalmente avaliar se o usuário NoBI é capaz de efetuar a mesma programação da SDN em controladores distintos, mas com um resultado operacional similar quando comparado àquele obtido a partir das interfaces nativas dos controladores, isto é, se a abordagem NoBI não fosse utilizada.

Em particular, optou-se por validar a NoBI considerando um cenário de acomodação de instruções de QoS na SDN, a fim de atender requisitos de transmissão de vídeo entre pontos remotos da rede. Isso porque, além de representar um exemplo real (controle de QoS é um tópico de grande interesse na área de Redes em geral), é especialmente adequado ao contexto atual, já que o provisionamento de QoS em redes SDN/*OpenFlow* não tem recebido a merecida atenção por parte da academia e indústria [Karakus e Durresi 2017]. Considerou-se aplicações de negócio que queiram transmitir vídeo entre dois pontos da rede SDN usando codificação H.264/MPEG-4 e HDR, no formato (1080p) e resolução (16:9), posto que este tipo de comunicação possui requisitos mínimos de transferência que devem ser atendidos. A fim de transmitir vídeo em alta velocidade com estas características de forma adequada, o experimento utiliza como parâmetro uma taxa de transmissão de 15 Mbps. Este é o mesmo limiar de referência que é recomendado pelo *YouTube* [Google, Inc. 2017] tendo em vista

assegurar a melhor experiência de visualização. Dessa forma, é possível avaliar o potencial de uso das funções NoBI para compatibilizar a troca de informações entre os diferentes controladores SDN e instalar as devidas instruções de QoS na rede visando atender tais requisitos de transmissão.

Adicionalmente, como a medição de parâmetros de QoS em ambientes SDN é uma tarefa difícil de ser realizada [Megyesi *et al.* 2017], a construção de uma prova de conceito com a implantação da NoBI sobre topologias SDN reais para a instauração de regras de QoS em planos de controle distintos pode contribuir para um maior avanço no desenvolvimento de métodos de avaliação de QoS em redes SDN em geral.

O restante deste capítulo é organizado da seguinte forma: na Seção 5.1, são expostos os objetivos definidos para este experimento. Em seguida, é apresentado o planejamento do experimento (Seção 5.2), que inclui os ambientes de testes funcionais utilizados nas avaliações (Subseção 5.2.1), as hipóteses iniciais estabelecidas (Subseção 5.2.2) e a identificação dos principais parâmetros que provocam impacto na variável de resposta (Subseção 5.2.3). Na Subseção 5.2.4, destaca-se o *design* de experimento escolhido, enquanto que a análise estatística dos resultados obtidos nas avaliações é levada a efeito na Seção 5.3. Por fim, algumas considerações finais do capítulo são apresentadas, na Seção 5.4.

5.1 Objetivos da Investigação

De maneira geral, o objetivo deste experimento é investigar: i) se o mesmo corpo de funções NoBI pode comunicar-se com diferentes controladores para programar a rede e atender uma necessidade da aplicação do negócio; e ii) caso o requisito de interoperabilidade seja satisfeito, é possível ainda observar um resultado operacional final similar àquele obtido a partir do uso das interfaces específicas dos controladores.

Em outras palavras, pode-se dizer que o experimento busca descobrir se há alguma diferença estatística significativa entre o uso da interface NoBI proposta para acomodar instruções na SDN e a outra abordagem tradicional, na qual o utilizador realiza a programação da rede usando as interfaces *northbound* estado da arte (os *"baselines"*) nativas dos controladores SDN selecionados, cujas abstrações são incompatíveis e portanto não podem ser utilizadas diretamente em um outro controlador.

De acordo com uma recomendação dos autores Wohlin *et al.* [Wohlin *et al.* 2012], o objetivo deste experimento pode ser descrito formalmente por meio do *framework* GQM (do inglês *Goal/Question/Metric*) [Basili e Rombach 1988; Solingen *et al.* 2002], como segue:

- ***analisar*** *diferentes técnicas para a programação da rede SDN (interface NoBI proposta, "baselines" estado da arte),* ***com o propósito de*** *compará-las,* ***com respeito à*** *reserva de largura de banda na transmissão de vídeo entre hosts,* ***do ponto de vista*** *do gerente de rede,* ***no contexto de*** *topologias de rede SDN reais e controladores distintos.*

5.2 Planejamento

Para este experimento, foi elaborado um planejamento formal, o qual é descrito nas seguintes subseções, como forma de sistematizar os procedimentos a serem realizados. É adequado frisar que as etapas deste plano, bem como a terminologia de experimentos utilizada ao longo deste capítulo, são baseadas em ensinamentos tradicionais encontrados na literatura; em particular [Jain 1991; Wohlin *et al.* 2012; Juristo e Moreno 2013].

5.2.1 Ambientes de Testes

Com o intuito de verificar o impacto das suposições de interoperabilidade da NoBI sobre os resultados produzidos pelos cenários de experimentação, foram implementadas duas arquiteturas de redes SDN completas, que incluem múltiplos *hosts* e os controladores *Floodlight* [Big Switch Networks 2013] e *OpenDaylight* [OpenDaylight 2013].

Particularmente, a principal diferença entre estas arquiteturas SDN reside no fato de que, na primeira, foi utilizado o *HP 5130 Brazil Switch–(JG975A)* [Hewlett-Packard 2017] como principal dispositivo comutador do plano de dados SDN. Este equipamento oferece suporte nativo ao protocolo *OpenFlow* versão 1.3. Na segunda topologia SDN, utilizou-se o *Zodiac FX* [Northbound Networks 2017], que também oferece suporte ao *OpenFlow* 1.3.

Buscou-se, portanto, não apenas analisar o comportamento da NoBI na comunicação direta e transparente com diferentes controladores, mas em redes SDN com diferentes planos de dados operando em uma nova versão *OpenFlow* diferente da v1.0, que foi a versão utilizada em um trabalho anterior [Vasconcelos *et al.* 2017].

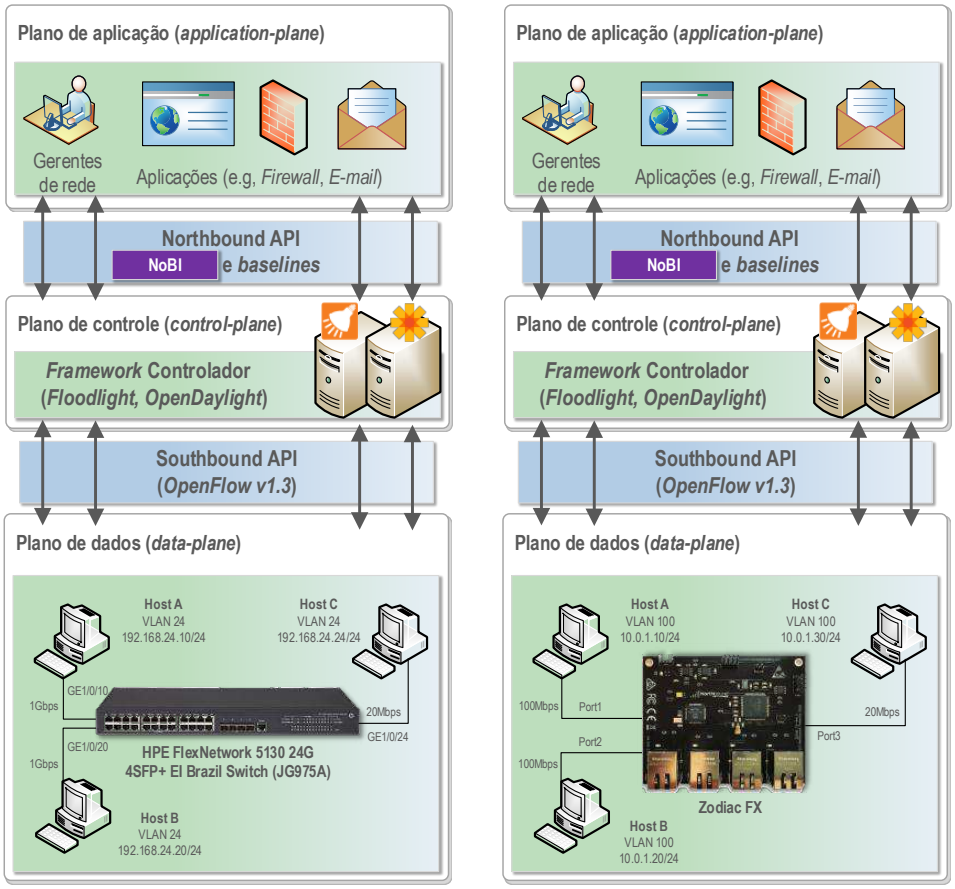
As configurações detalhadas dos componentes utilizados nos cenários de experimentação são apresentadas na Tabela 5.1.

Tabela 5.1: Detalhamento dos componentes utilizados nos ambientes de testes

Componente	**Descrição**
Controlador SDN	*Floodlight*: *release* 1.2, com suporte ao *OpenFlow* 1.0 e 1.3, imagem (compactada) de $51,1MB$ disponível em <http://github.com/floodlight/floodlight/tree/v1.2> e a dependência Java 7 (`openjdk-7-jdk`) instalada. *OpenDaylight*: *release* 0.4.4 *Beryllium-SR4*[a] com suporte ao *OpenFlow* v1.0 e v1.3, imagem (compactada) de $410MB$, Java 7 (`openjdk-7-jdk`) e *features* (`odl-l2switch-switch-ui`;`odl-dlux-all`) instaladas. Estes componentes foram executados em um computador com processador Intel(R) Core(TM) i7-3632QM 2.20 GHz, Memória RAM de 12GB, Placa de Rede Gigabit Ethernet e SO Debian Jessie 64 bits v8.7. Cada instância de controlador e a NoBI foram executadas em uma máquina virtual com SO Ubuntu 64 bits v14.04 LTS e Memória 4 GB RAM.
NoBI	As *engines FloodlightAdapterEngine* e *OpenDaylightAdapterEngine* foram instanciadas como objetos Java hospedados sobre o servidor de aplicação e contêiner de *Servlets* Apache/Tomcat v5.5.36, com o protocolo HTTP e bibliotecas XStream[b] e Gson[c] para o processamento de arquivos XML e JSON, respectivamente. Estes componentes foram executados em um computador com processador Intel(R) Core(TM) i7-3632QM 2.20 GHz, Memória RAM de 12GB, Placa de Rede Gigabit Ethernet e SO Debian Jessie 64 bits v8.7.
Computador Pessoal	Para fins de comunicação, foram usados três computadores cada um com processador Intel(R) Core(TM) i7-4790 3.60 GHz, Memória 4 GB RAM, Placa de Rede Gigabit Ethernet e SO Ubuntu 64 bits v14.04 LTS.
Equipamento OpenFlow	*HPE FlexNetwork 5130 24G 4SFP+ EI Brazil Switch*: gerenciável com 24 Portas 10/100/1000Mbps RJ-45, 4 Portas 1Gbps/10Gbps SFP fixas, Memória SDRAM de 1 GB, Número Serial BR58HCG086, Buffer de pacotes de 1,5 MB, Memória *flash* de 512 MB, Suporte ao *OpenFlow* 1.3 e *firmware* Comware Software v7.1.045. *Zodiac FX SDN Development Board*: 4 Portas Ethernet 100Mbps RJ-45, sendo 3 Portas *OpenFlow* v1.0/1.3, 1 Porta Ethernet para o Controlador, 1 Conector USB e Firmware v0.82.

[a] <*https://nexus.opendaylight.org/content/repositories/public/org/opendaylight/integration/distribution-karaf/0.4.4-Beryllium-SR4/*>
[b] <*https://x-stream.github.io/*>
[c] <*https://code.google.com/p/google-gson/*>

Essencialmente, cada uma das arquiteturas de rede SDN montadas para o experimento possui três *hosts*, um equipamento *OpenFlow* e o controlador de classe empresarial (*Floodlight* ou *OpenDaylight*). A organização geral destas topologias SDN implementadas é mostrada na Figura 5.1.



(a) (b)

Figura 5.1: Arquiteturas de redes SDN e equipamentos *OpenFlow* usados na avaliação: (a) HPE FlexNetwork 5130 JG975A e (b) Zodiac FX SDN Development Board

Os *hosts* A e B atuam como clientes, ao passo que o *host* C exerce o papel de servidor de *streaming* de vídeo. Observe que o enlace de conexão entre os *hosts*-clientes e o equipamento é de (a) 1 Gbps e (b) 100 Mbps, enquanto que o enlace com *host* C é de 20 Mbps. Dessa forma, foi possível estabelecer um cenário de congestionamento de tráfego, onde pacotes de *host* A deveriam ter prioridade sobre outros pacotes para receber um serviço diferenciado na rede. O gerente, portanto, deveria programar—via interfaces *northbound*—o encaminhamento do tráfego marcado de *host* A para uma fila de serviço, visando garantir uma reserva mínima de envio ao destino (*host* C). A ferramenta *iPerf* [Iperf 2008] foi empregada para produzir tráfego (e troca) de pacotes entre os nodos da rede, bem como computar a vazão resultante deste processo. Assim, amostras de dados puderam ser colhidas a partir da comunicação entre aplicações executadas em cada um dos *hosts*, a fim de examinar se houve alguma discrepância nas transmissões de vídeo obtidas a partir da utilização das diferentes técnicas (NoBI;*"baselines"*) para programar os requisitos de QoS na rede SDN.

5.2.2 Formulação de Hipóteses

A próxima etapa, após a definição dos principais objetivos do experimento e construção dos ambientes de testes, consistiu na formulação de hipóteses testáveis. Neste contexto, os autores Wohlin *et al.* [Wohlin *et al.* 2012] afirmam que a realização de testes de hipóteses formam a base da análise estatística de um experimento. Estes testes permitem verificar, com um elevado grau de confiança estatística, se as teorias e suposições iniciais nas quais o modelo de avaliação se baseia estão corretas.

Quanto à comparação entre as técnicas de programação da rede SDN, a questão se resume em saber se a NoBI é capaz de programar, em diferentes controladores SDN, um mesmo requisito de QoS com o propósito de assegurar velocidades (mínimas) de transmissão de vídeo na rede. Uma maneira de aferir a programação de QoS efetuada é medir as taxas de vazão resultantes das transmissões de vídeo na rede em cenário de congestionamento de tráfego. Ao ser reutilizado em diferentes controladores, o corpo de funções NoBI deve apresentar um resultado operacional similar àquele obtido da abordagem de configuração tradicional (chamada de *baseline*), que é feita a partir da interface nativa do controlador existente. No caso do *Floodlight*, as *baselines* consideradas são as interfaces nativas dos componentes *Static Entry Pusher* [Izard 2017] e *Floodlight Web GUI* [Izard 2017a]. Estes oferecem abstrações estado da arte para instalar instruções na rede e visualizar informações da SDN em tempo real em uma interface *Web*. No âmbito do controlador *OpenDaylight*, a *baseline* consiste na infraestrutura *Model-Driven Service Abstraction Layer* (MD-SAL) [OpenDaylight 2017b] em si, posto que ela disponibiliza ao gerente as principais APIs *northbound* para o controle da rede.

Isto posto, pôde-se definir a seguinte pergunta inicial:

- **Pergunta de pesquisa, $\mathbf{P_1}$**: existem reais diferenças entre as abordagens de programação da rede citadas (*baseline*, NoBI), considerando os resultados em termos da programação e reserva de recursos na rede para atender as transmissões de vídeo, ou as eventuais diferenças de resultado observadas nos cenários de experimentação são meramente atribuídas ao acaso?

Definição formal das hipóteses

Tomando como base a pergunta anterior, as seguintes hipóteses testáveis foram previamente formuladas:

- **Hipótese Nula, $\mathbf{H_0}$**: as técnicas para programação da rede SDN terão **o mesmo** desempenho operacional. Em outras palavras, as médias dos valores de vazão (μ_v) mínimos obtidos das técnicas (*baseline*, NoBI) **serão iguais**, considerando a acomodação de requisitos para a transmissão de vídeo usando a codificação H.264/MPEG-4 e HDR, formato (1080p) e resolução ($16{:}9$) na rede. Esta hipótese pode ser formalmente definida como sendo:

  $H_0 : \mu_v(baseline) = \mu_v(NoBI)$, ou
  $H_0 : \mu_d = \mu_v(baseline) - \mu_v(NoBI) = 0$, onde μ_d corresponde a diferença entre as médias de vazão mínimas.

- **Hipótese Alternativa, $\mathbf{H_A}$**: as médias de vazão mínimas obtidas a partir do uso das técnicas (*baseline*, NoBI) **não** serão iguais. Isto implica que elas dependem do tipo de abordagem para programação da rede utilizada. Ou ainda:

  $H_A : \mu_v(baseline) \neq \mu_v(NoBI)$, ou
  $H_A : \mu_d \neq 0$, onde μ_d corresponde a diferença entre as médias de vazão.

Apenas a análise estatística poderá refutar a hipótese H_0. Após os ensaios experimentais, entretanto, caso esta suposição de similaridade entre as técnicas se confirme, poder-se-á dizer com elevado grau de confiança que, além de poder ser reutilizado para orquestrar diferentes controladores, o corpo de funções NoBI é capaz de assegurar um resultado operacional equiparado aos *baselines* estado da arte disponíveis.

5.2.3 Fator e Variável de Resposta

Nesta seção, é apresentado o principal parâmetro (fator) de entrada do experimento, cujas alternativas serão sistematicamente manipuladas e controladas com o intuito de observar eventuais variações na métrica de desempenho escolhida.

- A variável independente (ou fator primário) neste experimento é a seguinte:

  - **Técnica de Programação da Rede**: neste experimento, haverá a variação controlada de duas abordagens: *baseline* e NoBI, no sentido de observar o efeito provocado por cada uma destas alternativas do fator na variável de resposta do experimento.

- Elencou-se a seguinte variável dependente (ou de resposta):

  - **Vazão de pacotes**: esta métrica de desempenho de rede corresponde à variável de resposta a ser observada. Neste estudo experimental, a média de vazão mínima (μ_v) é computada em Mbps.

Para efeito de orientação, o fator primário do experimento e seus respectivos níveis estão organizados na Tabela 5.2.

Tabela 5.2: Fator primário e seus respectivos níveis para DoE com 1 fator

Parâmetro de entrada (Fator)	**Valor (Nível)**
Técnica de Programação da SDN	$\{baseline; NoBI\}$

Na próxima seção, são apresentados detalhes acerca do *design* do experimento, bem como os passos que foram executados para a análise e interpretação confíável dos dados apurados.

5.2.4 Projeto do Experimento

Na seção anterior, viu-se que há apenas um fator de interesse (técnica de programação da rede SDN) com duas alternativas: $\{baseline; NoBI\}$. Estes níveis do fator foram atribuídos em cada ensaio experimental através do projeto de experimento (DoE, do inglês *Design of Experiment*) com 1 Fator (*One-factor Design*) [Jain 1991].

Com este DoE, é possível computar e comparar os efeitos provocados por cada nível do fator (técnica) na variável de resposta. Dessa forma, pode-se verificar, com um elevado grau de confiança, se a abordagem alternativa NoBI permite programar a rede SDN para atender às demandas da transmissão de vídeo com um desempenho operacional satisfatório, quando comparada às demais implementações nativas de diferentes controladores.

Plano de execução

Além de favorecer o surgimento de respostas confiáveis para o tema principal deste trabalho, o DoE citado estabelece um conjunto de passos que norteia o experimento de comprovação em si. Em particular, sua execução envolve as seguintes atividades:

- Coleta dos dados;
- Análise descritiva preliminar dos dados com gráficos de intervalos de confiança;
- Análise estatística inferencial para DoE com 1 fator e 2 alternativas $\{baseline; NoBI\}$; Para comparar os níveis do fator, os seguintes passos serão executados:
  - Identificação do modelo matemático a ser ajustado aos dados;
  - Computação dos efeitos de cada nível do fator na variável de resposta;
  - Estimação dos erros (resíduos) experimentais;
  - Alocação da variação causada pelas alternativas do fator e pelos resíduos;
  - Análise da variância e significância estatística dos efeitos provocados pelo fator;
  - Validação do modelo matemático;
  - Calculo dos intervalos de confiança para os efeitos provocados pelo fator;
- Se houver diferenças entre os efeitos, apontar qual, dentre os níveis do fator, é superior.

Procedimento

Para cada *host* cliente da SDN, computou-se uma vazão média, a partir de 50 (cinquenta) medições de vazão mínima coletadas na comunicação com o servidor (*host* `C`). Cada alternativa do fator técnica, i.e., $\{baseline; NoBI\}$, foi replicada 4 (quatro) vezes, visando contemplar cada tipo de controlador e equipamento. Dessa forma, foram necessários 8 (oito) ensaios experimentais, para obter médias com intervalos de confiança de 95%. Todos os níveis do fator foram aplicados a cada arquitetura SDN funcional, para que apenas a diferença entre as técnicas pudesse ser posteriormente examinada[1]. Este processo foi repetido até o final de

[1] Neste caso, diz-se que houve "pareamento". O pareamento é uma técnica que permite aumentar a precisão do experimento e o efeito da manipulação experimental das técnicas tornar-se-á mais aparente, já que uma eventual variação não-sistemática provocada pelo tipo do controlador/equipamento usado é reduzida.

todos os ensaios experimentais. A fim de minimizar algum tipo de viés, a alocação da técnica de programação à formação dos ensaios foi randomizada.

Neste experimento, decidiu-se, ainda, que todos os testes estatísticos seriam realizados com nível de significância estatística de 5% (ou nível de confiança de 95%). Particularmente, este valor típico determina a probabilidade, $\mathbb{P}$, de ocorrer um Erro do Tipo I, ou seja, $\mathbb{P}(rejeitar H_0 \mid H_0 = V)$. Em outras palavras, diz-se que há apenas 5% de chance de equivocadamente rejeitar a hipótese nula, no caso dela ser verdadeira.

Os detalhes sobre como cada atividade do corrente plano de execução foi levada a efeito são apresentados, na próxima seção.

5.3 Resultados e Análise

Concluído o planejamento do experimento, passou-se à etapa de coleta e análise dos dados.

Para assegurar a qualidade de visualização satisfatória nas transmissões feitas entre o servidor e *host* A (que receberia um serviço diferenciado na rede durante o congestionamento de tráfego), cada técnica aplicada buscava garantir ao cliente uma vazão mínima tão próxima quanto possível do valor de referência de 15 Mbps [Google, Inc. 2017]. Os resultados das medições realizadas em termos da média de vazão mínima (μ_v), em Mbps, são mostrados na Tabela 5.3.

Tabela 5.3: Médias de vazão mínima obtidas a partir das diferentes técnicas

		Técnicas de Programação da SDN		
Controlador	**Equipamento**	***baseline*** (μ_v)	**NoBI** (μ_v)	**Diferença**
Floodlight v1.2	HPE 5130	$15,26$	$15,21$	$0,33\%$
	Zodiac FX	$15,25$	$15,23$	$0,13\%$
OpenDaylight v0.4.4	HPE 5130	$15,20$	$15,20$	$0,00\%$
	Zodiac FX	$15,24$	$15,23$	$0,07\%$

De acordo com a tabela acima, os dados apurados sugerem que o uso da abordagem NoBI proposta é, por si só, suficiente para assegurar uma taxa mínima de transmissão de vídeo próxima da ideal (i.e., 15 Mbps). Adicionalmente, é possível constatar que as diferenças observadas entre as médias de vazão mínima (μ_v) obtidas das duas técnicas são discretas. Entretanto, para confirmar com maior segurança se os efeitos provocados pelas técnicas de

programação da SDN sobre a vazão são similares, uma análise (estatística) mais detalhada será levada a efeito, nas próximas seções.

5.3.1 Análise Descritiva Preliminar

Neste experimento, foi levada a efeito uma análise descritiva preliminar—momento em que foi produzido um gráfico com os intervalos de confiança robustos para as médias de vazão mínima computadas, com confiança estatística de 95%.

A partir das medições mostradas na Tabela 5.3 anterior, observou-se que as médias de vazão (μ_v) obtidas a partir do uso da técnica NoBI estão próximas do valor de referência de 15 Mbps e das taxas observadas com a outra técnica. Julgou-se, portanto, que nenhuma das abordagens utilizadas para programar a rede SDN parece ser superior à outra. Ao considerar o gráfico com os intervalos de confiança robustos (*bootstrap*), mostrado adiante na Figura 5.2, verifica-se que esta condição se confirma.

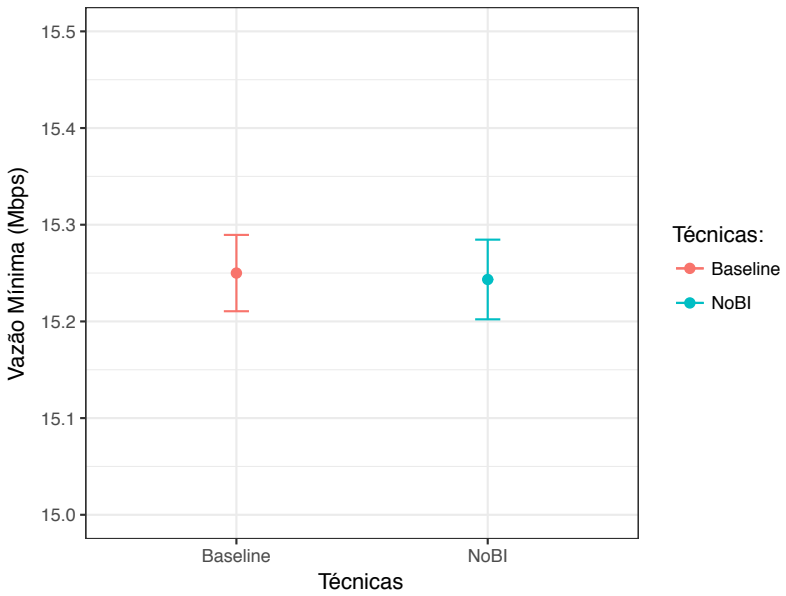


Figura 5.2: Gráfico de intervalos de confiança robustos (95%) computados para as médias de vazão mínima (μ_v). As técnicas *baseline* e NoBI foram aplicadas para estabelecer uma vazão mínima de 15 Mbps

Por exemplo, na Figura 5.2, pode-se observar que houve considerável interseção entre os intervalos de confiança, em ambas as técnicas. Este fato sugere que as médias de vazão produzidas após o emprego das técnicas NoBI e *baseline* não são estatisticamente diferentes. Em outras palavras, diz-se que há evidências preliminares de que não há diferenças significativas

entre o uso da interface NoBI e as abstrações de controle nativas dos controladores *Floodlight* e *OpenDaylight* para programar a rede SDN, nos diferentes equipamentos *OpenFlow HP 5130 Brazil Switch* [Hewlett-Packard 2017] e *Zodiac FX* [Northbound Networks 2017].

Com base no que foi apresentado na Figura 5.2 anterior, julgou-se ser adequada a utilização da técnica NoBI para programação da rede SDN tendo em vista o atendimento de demandas de transmissão de vídeo citadas no início deste capítulo. Entretanto, para confirmar esta conclusão (visual) preliminar de que as técnicas (*baseline*, NoBI) têm o mesmo desempenho operacional, as próximas subseções investigam o efeitos isolados de cada técnica na manutenção da vazão mínima, a estimação dos erros experimentais, a percentagem de variação explicada pelo fator na resposta, a análise de variância e, finalmente, o cálculo dos intervalos de confiança dos efeitos provocados pelos níveis do fator na vazão.

5.3.2 Identificação do Modelo Matemático

O modelo matemático empregado neste experimento para representar os dados amostrais obtidos das simulações é apresentado nesta seção.

De acordo com Jain [Jain 1991], o modelo matemático que deve ser ajustado (*fitted*) às amostras para a análise adequada dos dados coletados em um DoE com 1 Fator é descrito por meio da seguinte equação:

$$y_{ij} = \mu + \alpha_j + e_{ij} \tag{5.1}$$

onde, neste experimento, y_{ij} é a observação capturada na saída do experimento referente à i*-ésima* vazão média obtida com a aplicação da j*-ésima* alternativa do fator técnica, μ é a vazão média geral estimada pelo modelo, α_j é o efeito da alternativa j do fator na resposta e e_{ij} representa o erro residual inerente à observação em si. Ao longo das próximas seções, o leitor poderá observar que, além de estabelecer uma relação entre o fator e a resposta de interesse, este modelo permite avaliar a importância de cada alternativa do fator primário do experimento, com um elevado nível de confiança.

5.3.3 Computação dos Efeitos

Após a identificação do modelo matemático que representa as observações capturadas nas simulações, foi o momento de estimar o efeito isolado de cada alternativa do fator, i.e., $\{baseline; NoBI\}$, na variável de resposta.

Nesta ocasião, as seguintes equações foram utilizadas para tal finalidade [Jain 1991]:

$$\mu = \frac{1}{ar} \sum_{i=1}^{r} \sum_{j=1}^{a} y_{ij} \tag{5.2}$$

$$\bar{y}_{.j} = \frac{1}{r} \sum_{i=1}^{r} y_{ij} \tag{5.3}$$

onde, neste experimento, a é o número de alternativas do fator técnica, r é o número de replicações por alternativa, μ corresponde a vazão média geral estimada pelo modelo, $\bar{y}_{.j}$ é vazão média estimada para cada alternativa do fator e α_j é o efeito provocado pela alternativa j do fator na resposta. Os resultados da computação dos efeitos são ilustrados na Tabela 5.4.

Tabela 5.4: Cálculo dos efeitos das alternativas do fator

		Técnicas	
Controlador	**Equipamento**	***baseline***	**NoBI**
Floodlight v1.2	HPE 5130	$15,26$	$15,21$
	Zodiac FX	$15,25$	$15,23$
OpenDaylight v0.4.4	HPE 5130	$15,20$	$15,20$
	Zodiac FX	$15,24$	$15,23$
Média por alternativa ($\bar{y}_{.j}$)		$15,24$	$15,22$
Média geral estimada (μ)		$15,23$	$15,23$
Efeito da alternativa (α_j)		$0,01$	$-0,01$

A interpretação destes resultados é levada a efeito da seguinte forma: os efeitos computados para as técnicas *baseline* e NoBI são, respectivamente, $0,01$ e $-0,01$. Ou seja, o uso da técnica *baseline* para programar a SDN e atender às demandas de transmissão de vídeo provoca uma vazão $0,01$ Mbps maior do que a vazão média geral estimada pelo modelo matemático da Equação 5.1, que é de $15,23$ Mbps; ao passo que a vazão média obtida com a aplicação da técnica NoBI será $0,01$ Mbps menor que a média geral estimada pelo modelo.

5.3.4 Estimativa dos Erros Residuais

Nesta seção, o erros experimentais (resíduos) associados à execução do experimento serão examinados. Esta etapa é de suma importância, posto que a análise de resíduos é usada no processo de validação do modelo matemático ajustado aos dados amostrais (Subseção 5.3.6).

Observe que, ajustando-se a Equação 5.1 do modelo para $e_{ij} = y_{ij} - (\mu + \alpha_j)$, o erro residual e_{ij} de cada observação é dado pela diferença entre a vazão observada y_{ij} e o somatório entre a média geral μ estimada pelo modelo e o efeito α_j da alternativa j do fator. Tomando como base esta nova equação ajustada, o resíduo de cada observação capturada no experimento pode ser computado. A Tabela 5.5 abaixo ilustra os erros experimentais estimados pela equação do modelo matemático em cada medição realizada.

Tabela 5.5: Erros residuais associados às medições de vazão obtidas

		Técnicas	
Controlador	**Equipamento**	***baseline***	**NoBI**
Floodlight v1.2	HPE 5130	$0,02$	$-0,01$
	Zodiac FX	$0,01$	$0,01$
OpenDaylight v0.4.4	HPE 5130	$-0,04$	$-0,02$
	Zodiac FX	$0,00$	$0,01$

A partir desta tabela, observou-se que os valores dos resíduos são relativamente baixos. Esta condição sugere que a maior parte da variação na resposta (a vazão) se deve mais ao fator (técnica) do que aos erros experimentais. Entretanto, a confirmação deste resultado pela quantificação do impacto da técnica na resposta de interesse será apresentada na próxima seção, que trata do processo de alocação de variação.

5.3.5 Alocação da Variação

Nesta seção, é investigada a parcela de impacto do fator técnica na variável de resposta, ou seja, a repercussão do uso das alternativas *baseline* e $NoBI$ na obtenção de diferentes médias de vazão (μ_v) colhidas das transmissões de vídeo na rede SDN.

Primeiramente, considere que a variação total das médias de vazão obtidas com o uso das técnicas pode ser dividida em duas partes: a variação atribuída à técnica em si e aquela explicada pelos erros residuais. A partir daí, é possível quantificar e separar a parcela de

impacto do fator técnica (com suas duas alternativas) na obtenção de diferentes taxas de vazão, quando comparado ao efeito provocado pelos erros residuais neste mesmo processo.

Nesta ocasião, as seguintes equaçõqes foram utilizadas para tal finalidade [Jain 1991]:

$$SST = SSY - SS0 \tag{5.4}$$

$$SSY = SS0 + SSA + SSE \tag{5.5}$$

$$\sum_{i,j} y_{ij}^2 = \sum_{i=1}^{r}\sum_{j=1}^{a} \mu^2 + r\sum_{j=1}^{a} \alpha_j^2 + \sum_{i=1}^{r}\sum_{j=1}^{a} e_{ij}^2 \tag{5.6}$$

onde r é o número de replicações por alternativa, a é o número de alternativas do fator, SST é a variação total na variável de resposta, SSY é a soma dos quadrados das médias de vazão obtidas, $SS0$ é a soma dos quadrados das médias estimadas, SSA é a variação devida aos níveis do fator e SSE é a variação devida aos resíduos.

Neste experimento, foram obtidos os seguintes resultados:

- SSY $= \sum_{i,j} y_{ij}^2 = 1855,018$
- SS0 $= \sum_{i=1}^{r}\sum_{j=1}^{a} \mu^2 = 1855,014$
- SST $= SSY - SS0 = 0,004$
- SSA $= r\sum_{j=1}^{a} \alpha_j^2 = 0,001$
- SSE $= \sum_{i=1}^{r}\sum_{j=1}^{a} e_{ij}^2 = 0,003$

A percentagem de variação explicada pelo fator técnica segue a fórmula:

$$100 \times \frac{SSA}{SST} = 77,5\%$$

Já a percentagem de variação devida aos erros residuais é dada por:

$$100 \times \frac{SSE}{SST} = 22,5\%$$

Como se vê, a percentagem de variação nas médias de vazão explicada pelo fator técnica é de $77,5\%$. Por outro lado, a variação devida aos erros experimentais, aos equipamentos e aos

controladores foi de pouco mais de 22%. Neste contexto, os autores Juristo e Moreno [Juristo e Moreno 2013] destacam que a percentagem da variação total explicada pelo fator constitui uma medida efetiva para caracterizar sua *importância*. Diante destes resultados, concluiu-se, portanto, que a escolha da técnica para programar a rede SDN é o parâmetro de maior importância para a obtenção de diferentes medições de vazão.

5.3.6 Análise da Variância

Com o intuito de avaliar se a importância do fator técnica sobre erros residuais não ocorreu meramente devido ao acaso, um teste de significância estatística foi levado a efeito neste experimento e é apresentado nesta seção.

Primeiramente, considere r como sendo o número de replicações por alternativa e a como sendo o número de alternativas do fator técnica de programação. Os graus de liberdade para as somas dos quadrados SST, SSA e SSE são $ar - 1$, $a - 1$ e $a(r - 1)$, respectivamente. As médias quadráticas referentes ao fator técnica (MSA) e aos erros (MSE) são obtidas pela razão entre a soma dos quadrados correspondente e o respectivo grau de liberdade, ou seja, $MSA = \frac{SSA}{a-1}$ e $MSE = \frac{SSE}{a(r-1)}$. Finalmente, o F-Computado é dado pela divisão entre as médias quadráticas anteriores, isto é, $\frac{MSA}{MSE}$. Os resultados são apresentados na Tabela 5.6.

Tabela 5.6: Análise da significância dos efeitos das técnicas sobre os residuais

Componente	**Soma dos Quadrados**	**% de Variação**	**Graus de Liberdade**	**Média Quadrática**	**F-Computado**	**F-Valor (Tabela F)**
Taxas de vazão obtidas (μ_v)	SSY					
Média estimada	$SS0$					
Variação total	SST	$100,00$	7			
Técnicas (*baseline*, NoBI)	SSA	$77,50$	1	$MSA = 1 \times 10^{-3}$	$1,75$	$F\,[0,95; 1; 6] = 5,99$
Erros residuais	SSE	$22,50$	6	$MSE = 5 \times 10^{-4}$		

Como resultado da análise dos efeitos através do procedimento ANOVA (do inglês *Analysis Of Variance*) [Jain 1991], o F-Computado de $1,75$ e o F-Valor[2] de $5,99$ sugerem que não há diferenças significativas entre as técnicas (*baseline*, NoBI) sobre a variável de resposta. Já os percentuais de variação da Tabela 5.6 indicam que as técnicas de programação da rede tiveram contribuição de $77,50\%$ sobre a variação observada, enquanto que os controladores, equipamentos e resíduos apresentaram contribuição de $22,50\%$.

[2] F-Valor obtido de acordo com a Tabela A.7 do livro [Jain 1991].

Verificação dos Pressupostos Básicos do ANOVA

Para que conclusões obtidas até o momento sejam válidas, os pressupostos básicos do procedimento ANOVA [Jain 1991] devem ser atendidos. Estes são os seguintes:

- **Erros (resíduos) independentes**: para verificação dessa premissa, utilizou-se o teste *Durbin-Watson* [Durbin e Watson 1950]. Em particular, este teste permite detectar a presença de correlação (dependência) entre os erros residuais. Nesta ocasião, a hipótese nula a ser verificada é a de que os resíduos são independentes. Como resultado do teste, a estatística $\mathbb{D}$-$\mathbb{W}$ de $2,04$ e um *p-valor* de $0,10$ indicam que os erros associados às observações são independentes;

- **Erros normalmente distribuídos**: para investigar se os resíduos seguem uma Distribuição Normal, utilizou-se o teste *Shapiro-Wilk* [Shapiro e Wilk 1965]. Neste caso, a hipótese nula a ser verificada é a de que os resíduos são normalmente distribuídos. Ao aplicar este teste nos resíduos computados, obteve-se a estatística $\mathbb{W}$ de $0,52$ e um *p-valor* de $0,14$. Com estes resultados, julgou-se, portanto, que o teste não foi significativo (*p-valor* $> 0,05$) e que os erros residuais são normalmente distribuídos;

- **Erros com variâncias homogêneas (homocedasticidade)**: examinou-se esta premissa com o teste *Levene* [Levene 1960]. A hipótese nula considerada é a de que as variâncias de diferentes grupos de amostras são similares. Como resultado do teste, o *p-valor* de $0,98$ indica que os resíduos são homocedásticos.

A partir da confirmação[3] dos pressupostos básicos do ANOVA [Jain 1991], pôde-se dar andamento à análise com o objetivo de confirmar ou refutar a hipótese nula (Subseção 5.2.2) de que as técnicas (*baseline*, NoBI) têm o mesmo desempenho operacional.

5.3.7 Intervalos de Confiança dos Efeitos

O teste ANOVA [Jain 1991] executado na seção anterior não fornece indícios adicionais que permitam avaliar, com confiança, se uma abordagem é superior à outra ou não. Nesta

[3] Conforme discutido em [Field, Miles e Field 2012], há evidências de que o procedimento ANOVA seja robusto às violações de normalidade e homocedasticidade no processo de comparação dos grupos, quando "*os grupos possuem tamanhos iguais*" e "*os tamanhos das amostras são iguais*", respectivamente. Neste experimento, o tamanho de cada grupo comparado é 4 (quatro); e toda amostra obtida dos ensaios continha 50 (cinquenta) pontos de dados—o que reforça a robustez da análise.

seção, portanto, é apresentado um estudo complementar à análise da variância descrita na Subseção 5.3.6, com o objetivo de investigar se, de fato, há uma diferença significativa entre as técnicas (*baseline*, NoBI).

Uma vez que os parâmetros vazão geral média (μ) e o efeito (α_j) de cada técnica j na resposta, já calculados na Subseção 5.3.3, são variáveis aleatórias, é muito provável que outros valores sejam obtidos, no caso de novas simulações. Sendo assim, fez-se necessário computar os intervalos de confiança, para melhor representar os limites dentro dos quais os valores destes parâmetros do modelo matemático estarão contidos, em 95% dos casos. Nesta ocasião, as seguintes equações foram utilizadas [Jain 1991]:

$$s_e^2 = \sum e_{ij}^2 / \left[a(r-1)\right] \tag{5.7}$$

$$s_\mu = s_e / \sqrt{ar} \tag{5.8}$$

$$s_{\alpha_j} = s_e \sqrt{\left[(a-1)/ar\right]} \tag{5.9}$$

$$df_e = a(r-1) \tag{5.10}$$

onde s_e^2 é a variância dos erros residuais (e_{ij}), r é o número de replicações por alternativa, a é o número de alternativas do fator, s_μ é o desvio padrão do parâmetro μ, s_{α_j} é o desvio padrão do parâmetro α_j e df_e representa os graus de liberdade para os erros.

Os intervalos de confiança (95%) computados para os parâmetros vazão geral média (μ), efeito da técnica *baseline* (α_1) e efeito da técnica NoBI (α_2) foram, respectivamente, $\mu = (15,21; 15,25)$, $\alpha_1 = (-0,01; 0,03)$ e $\alpha_2 = (-0,03; 0,01)$. A interpretação desses intervalos é feita da seguinte forma: primeiro, pode-se afirmar que nenhuma das técnicas *baseline* (α_1) e NoBI (α_2) teve um efeito significativo sobre as diferenças de variação observadas nas taxas de vazão. Julgou-se, portanto, que em 95% dos casos, nenhuma das técnicas produzirá uma taxa de vazão *maior* ou *menor* do que aquela estimada pelo modelo matemático.

Para comparar as técnicas *baseline* e NoBI entre si, computou-se o intervalo de confiança do resultado da diferença entre os efeitos destas duas alternativas sobre a resposta de interesse,

isto é, $\alpha_1 - \alpha_2$. Para tal finalidade, as seguintes equações foram utilizadas [Jain 1991]:

$$\mu_{\alpha_1-\alpha_2} = \bar{y_{.1}} - \bar{y_{.2}} \tag{5.11}$$

$$s_{\alpha_1-\alpha_2} = \frac{s_e}{\sqrt{\sum h_j^2/ar}} \tag{5.12}$$

onde $\mu_{\alpha_1-\alpha_2}$ é a diferença entre as médias de vazão estimadas pelo modelo para cada técnica ($\bar{y}_{.j}$) e $s_{\alpha_1-\alpha_2}$ é o desvio padrão para a diferença $\alpha_1 - \alpha_2$.

Nesta etapa, o intervalo de confiança computado para $\alpha_1 - \alpha_2 = (-0,08; 0,12)$. Como este intervalo resultante da diferença $\alpha_1 - \alpha_2$ inclui o valor 0 (zero), pode-se concluir que: i) $\alpha_1 = \alpha_2$, ou seja, não há diferença entre os efeitos provocados pela aplicação das técnicas *baseline* e NoBI na reserva de largura de banda; e ii) nenhuma das técnicas de programação da SDN é superior à outra. Em outras palavras, isso implica dizer que o corpo de funções NoBI pode programar a SDN com um desempenho operacional equiparado às abstrações das interfaces nativas dos controladores, em 95% dos casos[4].

Como resultado de $\alpha_1 - \alpha_2$, a hipótese nula (H_0) inicial da Subseção 5.2.2 de que as duas técnicas de programação da rede SDN terão o mesmo desempenho operacional, ou seja, as medições de vazão obtidas independem da abordagem utilizada, não pode ser refutada, com confiança estatística de 95%. Confirma-se, portanto, a premissa de que o efeito provocado pelo emprego da técnica NoBI provoca médias de vazão similares àquelas induzidas pelas implementações *northbound* estado da arte oferecidas pelos controladores *Floodlight* e *OpenDaylight*, em 95% dos casos.

5.3.8 Testes de Carga

Quanto à sobrecarga de esforço envolvido na identificação, tradução dinâmica e instalação de instruções no controlador existente, de fato, a utilização do corpo de funcionalidades NoBI representa uma camada adicional de abstração que pode incorrer em sobrecarga (*overhead*).

Em um teste de carga levado a efeito com a ferramenta *Apache JMeter* [ASF 2018], os tempos de resposta médios (τ_r) (em ms) foram computados a partir várias requisições HTTP feitas com as técnicas $\{baseline; NoBI\}$. Os resultados obtidos são mostrados na Figura 5.3.

[4] Este resultado confirma as evidências visuais preliminares da análise descritiva da Subseção 5.3.1.

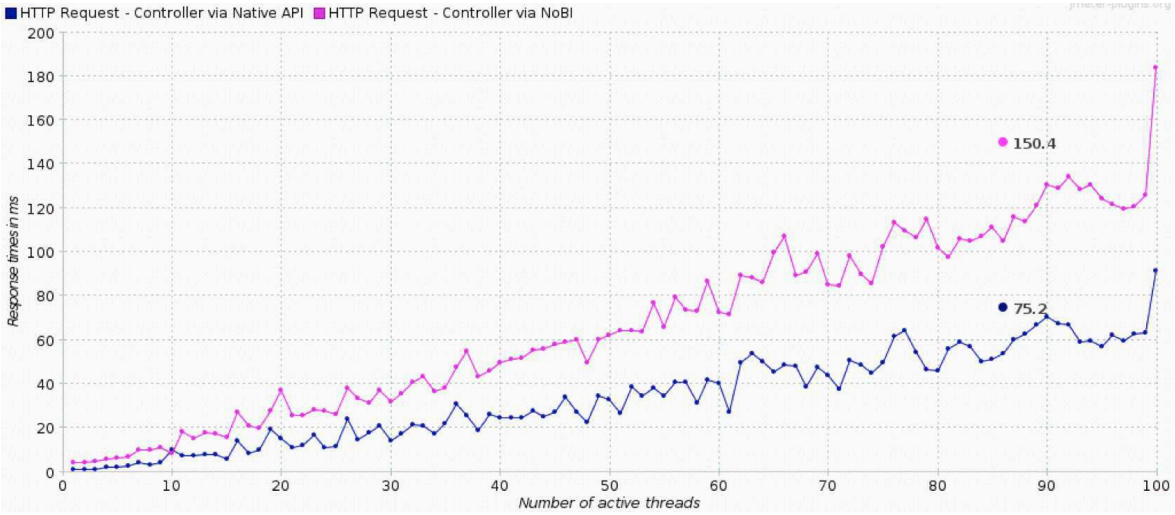


Figura 5.3: Tempos de respostas médios computados da comparação das abordagens

A fim de analisar o comportamento da NoBI sob alta demanda de acesso, a carga de trabalho utilizada teve como objetivo estressar a disponibilidade para instalar instruções na SDN e a concorrência pelo uso. Para isso, uma centena de usuários foi instanciada no *JMeter* e um total de 70×10^3 requisições foram feitas em paralelo às interfaces NoBI e nativa do controlador. Com a abordagem NoBI proposta, o τ_r final computado pela ferramenta foi de $150,4$ ms atingindo um pico de resposta pouco acima de 180 ms, enquanto que a mesma carga de trabalho aplicada à interface nativa do controlador resultou em um τ_r de $75,2$ ms com pico de resposta abaixo de 100 ms. Nesta ocasião, julgou-se que, com a abordagem NoBI, um aumento no tempo de resposta médio foi observado, mas inferior à 76 ms sob uma carga de 70×10^3 acessos simultâneos.

5.3.9 Análise Qualitativa

Além da implantação da NoBI em um ambiente SDN real para observar o comportamento de seu corpo de funções, é recomendado que a abordagem proposta possa ser submetida a uma avaliação qualitativa. Dessa forma, pode-se obter uma participação e *feedback* de usuários reais, tendo em vista verificar a aceitação e coletar a opinião geral de uso da interface NoBI, quando comparada às interfaces nativas oferecidas pelos controladores. A ideia, portanto, é que usuários pudessem se posicionar acerca da facilidade de uso das funções NoBI produzidas e dos benefícios da interoperabilidade de interfaces *northbound*.

O público-alvo desta análise consistiu de docentes e profissionais da área de Redes de Computadores dos setores público, acadêmico e privado, além de alunos de graduação e pós-graduação com alguma experiência em SDN. A abordagem utilizada para coleta dos dados constistiu na aplicação de um questionário (vide Apêndice B)[5] com perguntas objetivas.

No Apêndice B, observe que a seção inicial do questionário se subdividiu em perguntas que tratavam do setor de trabalho e o tempo de experiência do participante na área de SDN. As etapas seguintes do questionário, por outro lado, buscavam avaliar a facilidade de uso da abordagem NoBI. Neste momento, eram apresentadas questões que tratavam da facilidade de uso das funcionalidades da NoBI, quando comparada àquelas encontradas nas interfaces nativas dos controladores *Floodlight* e *OpenDaylight*, na execução de atividades na rede, tais como: a) a consulta de informações da topologia; b) listagem das regras de controle de fluxo de pacotes ativas na rede; c) obtenção de informações dos equipamento da rede; d) encaminhamento de pacotes ao controlador; e e) encaminhamento de tráfego marcado prioritário para filas de serviço e reserva de largura de banda mínima. Para cada atividade, o participante deveria apontar, em sua opinião, a interface cuja funcionalidade fosse mais intuitiva e simples de utilizar. Ainda, em cada atividade, foram apresentados 3 tipos de alternativas: a execução da atividade com a função da NoBI, a mesma execução com uma função nativa da interface do controlador e, por fim, se as interfaces teriam a mesma facilidade de uso (isto é, não havia diferenças ou "neutro").

Um total de 18 participantes respondeu todo o questionário. Destes, 14 declararam ter experiência em SDN nos ambientes acadêmico e corporativo, enquanto que os demais declararam ter conhecimento do assunto, mas com pouca experiência. Aproximadamente 60% dos participantes tinham experiência entre 1 e 3 anos, e mais de 6 anos com a área redes SDN, enquanto que os demais declararam ter menos de 1 ano de experiência. Os dados apurados são mostrados na Tabela 5.7. Com base em tais dados, foi gerado ainda um gráfico de facilidade de uso na execução das atividades na rede SDN, conforme pode-se ver na Figura 5.4.

Os resultados dos dados apurados mostram que, na opinião dos 18 participantes, dentro das cinco atividades de operação na SDN apresentadas, a interface NoBI foi a que apresentou maior facilidade de uso. Em particular, em um somatório realizado das frequências de respostas, obtêm-se o valor de 72 favoráveis à NoBI sobre 90 observações colhidas no total.

[5] <https://goo.gl/forms/J2K7CUl5JUBBuPhx2>

Tabela 5.7: Indicação de Facilidade de Uso da Interface para Execução de Atividades

Atividade	**NoBI**	**Interface do Controlador**	**Neutro**
Consulta da topologia	14	2	2
Listagem de regras ativas	13	4	1
Consulta de switches	15	2	1
Envio de pacotes ao Controlador	14	4	0
Reserva de banda	16	1	1
Frequência de respostas	**72**	**13**	**5**

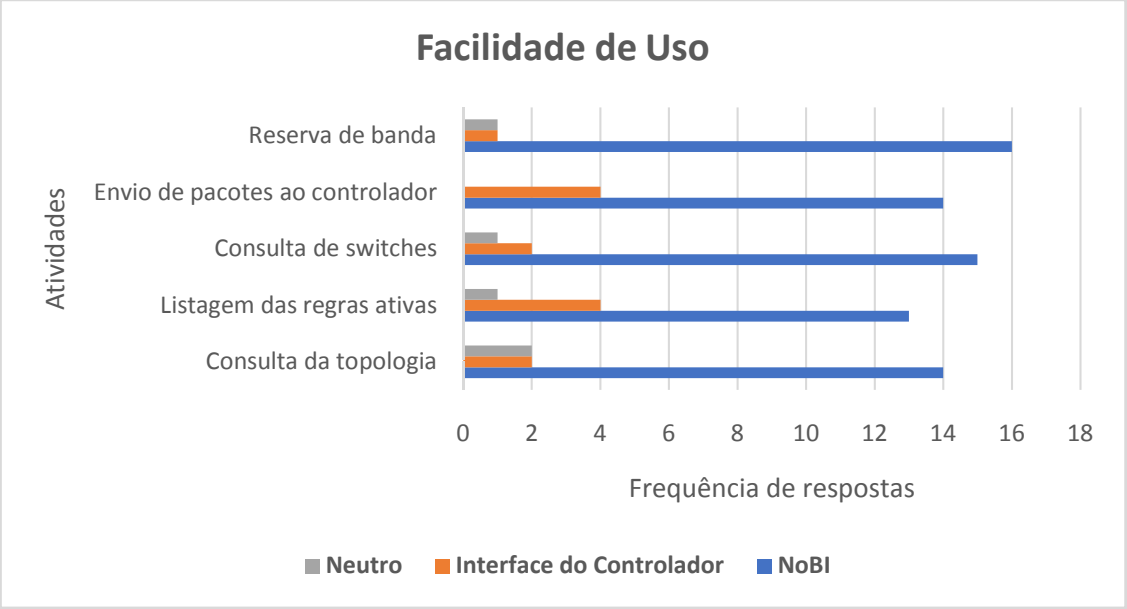


Figura 5.4: Facilidade de Uso na Execução de Atividades

Em outras palavras, pode-se dizer que 80% das respostas foram favoráveis à NoBI, enquanto que 14% foram favoráveis à interface do controlador e apenas 6% das respostas dos usuários apontaram que as interfaces possuíam a mesma facilidade de uso.

Uma vez que a proporção observada ($\hat{p}$) de 80% é uma variável aleatória, é muito provável que outros valores sejam obtidos, no caso de novas coletas. Sendo assim, fez-se necessário computar os intervalos de confiança, para melhor representar os limites dentro dos quais os valores da proporção real ($\bar{p}$) estará contida, em 95% dos casos. A hipótese nula considerada nesta etapa foi a de que *menos* de 50% da população dos usuários seria favorável à NoBI.

O intervalo de confiança (95%) computado para o parâmetro ($\bar{p}$) foi de $\bar{p} = (0,70; 0,87)$.

Julgou-se, portanto, que mais da metade dos usuários (70-87%) será favorável à NoBI, em vez das interfaces nativas dos controladores, em termos de facilidade de uso, na maioria dos casos.

Na última pergunta do questionário, os participantes eram indagados sobre os ganhos de interfaces com suporte à interoperabilidade entre controladores distintos para o gerente de redes. Nesta questão, os participantes foram unânimes em referendar a importância do desenvolvimento de interfaces mais flexíveis, de corpo funcional uniforme e capazes de se comunicar de forma transparente com diferentes controladores SDN.

5.3.10 Discussão

Resumidamente, as principais conclusões alcançadas com a análise estatística e o significado dos resultados apresentados até o momento são descritos a seguir:

- primeiramente, viu-se que o mesmo corpo de funções da NoBI foi capaz de comunicar-se com subsistemas de controladores *OpenFlow* completamente distintos, orquestrando-os para acomodar instruções na rede SDN, de forma automática;

- ao examinar o comportamento da interface NoBI junto aos *frameworks Floodlight* e *OpenDaylight*, e em diferentes equipamentos *OpenFlow*, viu-se que ela apresentou um resultado operacional equiparado àquele obtido da abordagem de configuração tradicional (chamada de *baseline*) da SDN, feita a partir das implementações estado da arte [Izard 2017; OpenDaylight 2017b] destes controladores;

- em particular, com o mesmo corpo de funções da NoBI, pôde-se acomodar requisitos de QoS na rede independentemente do controlador existente—*Floodlight* ou *OpenDaylight*. Isso implica dizer que o modelo de dados agnóstico a estes controladores SDN (Capítulo 4) e usado para representar os parâmetros de controle da rede mostrou-se útil;

- um estudo adicional para investigar a escalabilidade da abordagem proposta, no que diz respeito ao número de conexões concorrentes sob demanda, mostrou que a NoBI foi capaz de suportar uma carga crescente de trabalho de maneira uniforme e apresentando um tempo de resposta médio satisfatório quando submetida a um cenário desafiador envolvendo milhares de acessos simultâneos;

- sob uma perspectiva estatística, a suposição de similaridade entre as abordagens tradicional e NoBI para programar a rede se confirmou, com um elevado grau de confiança (Subseção 5.3.7). Uma análise inferencial realizada aponta que a programação de instruções de QoS na rede com a interface NoBI provocará taxas de transmissão similares àquelas obtidas pela abordagem padrão de configuração que é oferecida pelo controlador, em 95% dos casos. A repercussão do uso das técnicas comparadas na obtenção das amostras (Subseção 5.3.5) superou os efeitos causados por erros residuais (Subseção 5.3.4) inerentes ao experimento em si. Ainda, viu-se que a importância das técnicas estudadas sobre os resíduos não ocorreu meramente devido ao acaso (Subseção 5.3.6);

- em uma análise qualitativa levada a efeito, foi reportado que o corpo de funções é mais intuitivo e simples de utilizar, quando comparado às implementações disponíveis pelos controladores;

- de um modo geral, pode-se dizer que a construção de uma prova de conceito com a implantação da NoBI sobre arquiteturas de rede SDN reais e uma avaliação estatística do seu desempenho mostram não apenas a viabilidade da solução desenvolvida nesta tese, como também o fato de que ela pode representar um caminho promissor.

5.3.11 Ameaças à Validade

Nesta seção, são discutidas as principais ameaças à validade deste estudo, à luz da taxonomia apresentada em [Cook, Campbell e Day 1979]. São elas:

- primeiramente, é importante frisar que os resultados da análise estatística foram examinados a partir de arquiteturas SDN contendo um controlador para orquestrar os equipamentos do plano de dados. Porém, há outras categorias de redes SDN, conhecidas como SDN multi-controladas [Blial, Mamoun e Benaini 2016]. Esta condição dificulta a generalização das conclusões obtidas sobre estes tipos particulares de redes, os quais podem conter vários controladores coordenados. Portanto, tomando como base o princípio da *validade externa* [Cook, Campbell e Day 1979], acredita-se ser um desafio observar as mesmas evidências estatísticas sobre uma população maior de redes SDN com características tão distintas das arquiteturas SDN (mono-controladas) utilizadas neste estudo;

- uma outra ameaça à validade externa diz respeito ao versionamento do protocolo *southbound*. Particularmente, muitas das estruturas de programação nativas dos controladores dependem de versões específicas do protocolo *OpenFlow*, o que dificulta a portabilidade das funções NoBI. Em um trabalho anterior [Vasconcelos *et al.* 2017], analisou-se o comportamento da NoBI em redes *OpenFlow* v1.0, enquanto que esta avaliação experimental utilizou a versão v1.3. Dessa forma, não se pode afirmar com segurança que as conclusões obtidas podem ser generalizadas para outras redes SDN, além das que podem operar nas versões *OpenFlow* v1.0 e v1.3;

- uma ameaça de *constructo* diz respeito a métrica usada para avaliar a interoperabilidade da interface entre controladores distintos. Primeiramente, é importante frisar que não há uma única métrica padrão para avaliar a interoperabilidade de uma interface *northbound*. Neste estudo, como a programação foi feita em termos da instalação de instruções de QoS com a NoBI tendo em vista estabelecer uma reserva de largura de banda mínima em diferentes controladores-alvo, escolheu-se a vazão como métrica para verificar o resultado operacional final após a programação. Entretanto, dependendo do cenário estabelecido, outros tipos de programação poderiam ser realizados, da mesma forma que outras métricas poderiam, pois, ser mais adequadas. Dessa forma, a métrica utilizada neste experimento para avaliar o resultado final da programação da SDN com a NoBI nos diferentes controladores não pode ser generalizada como a mais adequada para outros estudos experimentais com metas, características e cenário variados.

5.4 Considerações Finais

Neste capítulo, foi apresentada uma proposta particular de validação da abordagem alternativa para programar a rede SDN com foco na interoperabilidade e maior independência do controlador: a NoBI.

Para observar o comportamento e demonstrar a viabilidade do corpo de funções interoperável NoBI, a avaliação usou arquiteturas de redes SDN contendo diferentes equipamentos *OpenFlow* reais, além de controladores SDN de classe empresarial com mecanismos de controle completamente distintos. Sendo assim, o autor da tese acredita que o *testbed* funcional mostrado neste capítulo se mostrou útil para implantar e avaliar a abordagem proposta.

Particularmente, investigou-se o potencial de uso da abordagem NoBI para acomodar instruções de QoS na SDN dinamicamente e atender requisitos de aplicações que operam com a transmissão de vídeo entre pontos remotos. No entanto, é perfeitamente viável encontrar outras categorias de aplicações que poderiam se beneficiar do conjunto de funcionalidades da NoBI, a exemplo de ambientes de execução de jogos com múltiplos jogadores.

Para tecer conclusões relativas à programação da rede de forma confiável, usou-se técnicas como *design* de experimento, para mensurar e comparar os efeitos provocados por cada parâmetro experimental na variável de resposta de interesse.

A análise dos resultados aponta que a abordagem proposta neste trabalho permite ao gerente utilizar uma mesma interface de configuração para orquestrar redes SDN com planos de controle diferentes, apresentando um desempenho operacional similar àquele obtido a partir das interfaces específicas dos controladores, as quais não podem ser utilizadas pelo gerente diretamente em outro controlador. Dessa forma, o estudo experimental mostra a viabilidade da NoBI, como também o fato de que ela pode representar um caminho promissor, pois baseia-se na programação da rede através de um corpo de funções que pode ser reutilizado pelo gerente sem esforço e na simplificação das tarefas tradicionais de escrita e instalação (manual) de regras de controle por parte deste profissional sobre o controlador existente.

Considerando a detecção de eventuais perdas de desempenho envolvidas, observou-se um aumento do tempo de resposta médio computado, quando a programação da SDN era realizada com a interface NoBI. No entanto, esta perda deve-se tão somente aos diversos processos automáticos de tradução e composição das regras de controle feitos pelas *engines* NoBI, bem como a comunicação e instalação das instruções produzidas em tempo de execução nos subsistemas do controlador existente. Entretanto, a NoBI mostrou-se intuitiva e mais simples de utilizar do que as implementações nativas dos controladores, de acordo com os resultados obtidos da análise qualitativa realizada.

É importante destacar que a hipótese preliminar de similaridade entre as técnicas NoBI e *baseline* representava o que havia sido detectado no ambiente estudado. No entanto, uma análise estatística inferencial realizada aponta que a utilização da interface NoBI provocará resultados operacionais similares àqueles obtidos a partir da abordagem padrão de configuração que é oferecida pelo controlador existente, na grande maioria dos casos.

Capítulo 6

Conclusão

Este trabalho de doutorado abordou o problema de suporte à portabilidade (reutilização) de abstrações de controle das interfaces *northbound* para a programação de redes definidas por *software*. A principal razão para escolha deste tema é o impacto positivo de interfaces mais flexíveis e agnósticas ao controlador sobre a redução do tempo e complexidade para elaborar e instalar instruções nestas redes. O impacto destas interfaces é ainda maior em redes SDN multi-controladas [Blial, Mamoun e Benaini 2016], onde gerentes de redes precisam lidar com controladores cujas interfaces, linguagens, instruções e modelos de representação de dados são completamente distintos e incompatíveis.

Diversas pesquisas foram examinadas nesta tese, com vistas à identificar formas alternativas de suporte à portabilidade da programação da SDN. Os estudos tiveram por base o levantamento do estado da arte de pesquisas realizadas nos últimos anos na área, que tratam da programação de redes SDN usando interfaces *northbound*. Particularmente, observou-se que trabalhos anteriores propuseram soluções úteis e que permitem ao gerente modificar o comportamento da rede, mas que não endereçam questões de reuso do corpo funcional da interface e sua interoperabilidade ante a planos de controle SDN distintos. O resultado é que a dependência de uma plataforma particular de *software* eleva custos operacionais da rede, reduz a produtividade e dificulta a inovação [Masoudi e Ghaffari 2016].

Conforme foi discutido neste documento, no âmbito de controladores SDN de classe empresarial disponíveis, a ampla diversidade de implementações e a ausência de um consenso quanto à padronização de suas interfaces *northbound* de controle resultam em múltiplas formas de se programar a rede, acarretando complexidade adicional para os gerentes.

Diante deste problema, foi desenvolvida nesta tese uma abordagem alternativa para programação da SDN em alto nível, denominada `NoBI`. Em particular, trata-se de uma interface *northbound* concebida visando fornecer aos gerentes destas redes uma camada de abstração com suporte à comunicação transparente com mais de um controlador, que prioriza a acomodação simplificada e automatizada de instruções na rede, em tempo de execução, a partir de um corpo de funcionalidades uniforme e que pode ser reutilizado sem esforço. Com a `NoBI`, não é preciso injetar comandos específicos de controladores diretamente nas regras de controle da rede, a fim de ampliar a portabilidade de instruções e o suporte a múltiplos controladores—todo o esforço do gerente para tratar a incompatibilidade e variabilidade entre as interfaces nativas dos controladores é reduzido e transferido para a `NoBI`.

A seguir, apresenta-se as principais contribuições do presente trabalho e perspectivas de trabalhos futuros.

6.1 Contribuições

A partir dos resultados obtidos neste trabalho, pode-se destacar as seguintes contribuições:

- **Desenvolvimento de uma API para programação da SDN –** foi desenvolvida uma interface *northbound* para programar redes SDN, chamada NoBI, que pode ser considerada uma alternativa ao projeto convencional destas interfaces. O principal diferencial da solução proposta neste trabalho é o foco na independência de controlador e a portabilidade (reuso) de seu corpo de funcionalidades.

- **Definição de um arcabouço de projeto extensível –** foi definido um projeto de classes baseado em padrões de projeto, de modo a ser flexível e poder acomodar outras entidades para oferecer suporte a novos controladores com facilidade. Diferentemente do projeto de alguns controladores SDN, o *design* do corpo de funções NoBI não viola princípios arquiteturais REST [Fielding *et al.* 1999].

- **Construção de modelos de dados –** foram desenvolvidos modelos de dados unificados, para representar os parâmetros de controle da rede, uma vez que cada controlador possui suas próprias estruturas de dados e formatos de arquivos. Em particular, os modelos definidos neste trabalho permitem ao gerente trabalhar com um conjunto genérico

e padronizado de instruções, ao invés do aprendizado de comandos específicos dos controladores SDN ou a reelaboração e compatibilização de instruções existentes.

- **Desenvolvimento de *engines* de tradução –** foram projetadas e implementadas neste trabalho *engines Java* capazes de traduzir as URIs e parâmetros de controle da rede usados pelo gerente em instruções específicas do controlador-alvo. Tais *engines* foram compatibilizadas com subsistemas nativos dos controladores e mapeiam as estruturas do modelo de dados NoBI unificado em modelos correspondentes suportados pelos controladores *Floodlight* e *OpenDaylight*, de forma automática.

- Avaliação da viabilidade de interfaces *northbound* reutilizáveis e focadas na portabilidade, usando arquiteturas de redes SDN reais com bons representantes existentes de controladores [Big Switch Networks 2013; OpenDaylight 2013] e ainda diferentes equipamentos de rede [Hewlett-Packard 2017; Northbound Networks 2017] com suporte ao *OpenFlow*. Foram realizados experimentos que apontaram que a solução apresentada neste trabalho se mostrou mais eficiente e flexível, em termos de portabilidade, do que outras interfaces para programação da rede estudadas. Isso porque o mesmo corpo de funções `NoBI` pode ser reutilizado para orquestrar planos de controle SDN completamente distintos, e com um resultado operacional similar àquele obtido a partir das diferentes interfaces estado da arte oferecidas pelos controladores.

- Diferentemente de vários trabalhos relacionados, os quais não fornecem detalhes suficientes quanto aos protótipos das abstrações de controle desenvolvidas ou como estas devem ser efetivamente usadas para programar a SDN, esta tese apresenta detalhes de utilização de todas as funções implementadas. Em particular, discute-se como as passagens de parâmetros devem ser realizadas pelo gerente e diversos exemplos de retorno de código são mostrados em mais de um controlador.

- Análise do impacto da localização do controlador na rede SDN sobre o desempenho de comunicação entre *hosts* de uma rede SDN [Vasconcelos *et al.* 2014].

- Investigação dos principais obstáculos enfrentados pelos gerentes na programação de redes SDN, bem como a identificação dos benefícios da busca por interfaces mais flexíveis, focadas na reutilização e portabilidade de abstrações de controle, além do

impacto positivo destas interfaces para a consolidação das lojas virtuais de aplicativos para redes SDN [Vasconcelos *et al.* 2015].

- Introdução dos resultados e conhecimentos obtidos nesta tese na literatura de Redes Definidas por *Software* por meio das seguintes publicações:

  – VASCONCELOS, R. C. *et al*. Evaluating the Geographic Distance Effects in Communication Between Hosts and a Controller on a Software-defined Network. In: *Proceedings of the 13th International Conference WWW/INTERNET 2014 (ICWI)*. Porto, Portugal: IADIS, 2014. p. 275–282. ISBN 978-989-8533-24-1.

  – VASCONCELOS, R. C. *et al*. Towards an SDN-App Store: A Generic Northbound API for Software-Defined Networks. In: *Proceedings of the 14th International Conference WWW/INTERNET 2015 (ICWI)*. Maynooth, Dublin, Ireland: IADIS, 2015. p. 173–176. ISBN 978-989-8533-44-9.

  – VASCONCELOS, C. R. *et al*. Enabling High-Level Network Programming: A Northbound API for Software-Defined Networks. In: *31st International Conference on Information Networking* (ICOIN 2017). Da Nang, Vietnam: IEEE, 2017. p.662–667.

6.2 Trabalhos Futuros

As contribuições anteriores apresentadas permitem identificar alguns rumos futuros, os quais podem ser tomados para dar continuidade à pesquisa desenvolvida nesta tese. Abaixo, são apresentadas algumas recomendações para trabalhos futuros:

- uma vez que as interfaces *northbound* constituem um dos principais elementos em redes SDN, pode-se realizar estudos com vistas à criação de novas funcionalidades e estruturas de representação dados de controle da SDN voltados para domínios de problemas específicos não abordados por este trabalho (e.g., abstrações de controle e novos modelos unificados para programação de redes com foco na segurança). Dessa forma, a inclusão de outras funções e modelos de dados abrirá novos caminhos em direção a uma maior automatização, reutilização de instruções e inovação em SDN.

- outro trabalho interessante que pode ser realizado é a análise e incorporação de funcionalidades do protocolo *OF-Config* (do inglês *OpenFlow Management and Configuration Protocol*) [ONF 2014a] à solução proposta. Em particular, este protocolo prevê capacidades de gerenciamento que irão complementar aquelas previstas pelo *OpenFlow*. Entretanto, o *OF-Config* se encontra em processo de padronização e homologação por parte da ONF [Marschke, Doyle e Moyer 2015]. Pretende-se, com isto, evoluir um pouco mais em termos de configuração da rede em tempo de execução, além da busca por novas formas de se gerenciar os equipamentos *OpenFlow*.

- vários trabalhos existentes na literatura de referência têm conduzido experimentos em redes SDN contendo um plano de controle. Pode-se, porém, propor estudos que investiguem o comportamento de interfaces reutilizáveis, mas em arquiteturas SDN com controladores distribuídos. Para isto, será necessária elaboração de novos experimentos que observem não apenas a comunicação com múltiplos controladores distintos, mas, sobretudo, o uso de abstrações *east/westbound* reutilizáveis para sincronização, processamento de eventos e transferência de dados envolvendo estas partes. Essa abordagem ainda não existe na literatura de Redes SDN.

- pode-se investigar, ainda, estratégias para reduzir a heterogeneidade de instruções de diferentes versões do protocolos *OpenFlow* e, a partir disto, propor uma versão derivada da abordagem NoBI, tendo em vista ampliar o suporte nativo a outras versões existentes.

- é interessante implantar a interface proposta em um ambiente externo à academia – na indústria, de preferência. Uma vez que o experimento de validação definido neste trabalho foi projetado e instanciado no ambiente acadêmico, pode-se investigar, por exemplo, a aceitação geral da interface por parte de gerentes em SDNs de grande porte e até de desenvolvedores de aplicações de *software* voltadas para ambientes SDN.

Bibliografia

[Akyildiz *et al.* 2014] AKYILDIZ, I. F. *et al.* A Roadmap for Traffic Engineering in SDN-OpenFlow Networks. *Computer Networks*, Elsevier North-Holland, Inc., New York, NY, USA, v. 71, p. 1–30, October 2014. ISSN 1389-1286.

[Arbettu *et al.* 2016] ARBETTU, R. K. *et al.* Security Analysis of OpenDaylight, ONOS, Rosemary and Ryu SDN Controllers. In: *17th International Telecommunications Network Strategy and Planning Symposium (Networks)*. Montreal, QC, Canada: IEEE, 2016. p. 37–44.

[ASF 2018] ASF. *Apache Software Foundation:JMeter*. 2018. Disponível em: <http://jmeter.apache.org/>.

[Azodolmolky 2013] AZODOLMOLKY, S. *Software Defined Networking with OpenFlow*. 1st. ed. Birmingham, UK: Packt Publishing Ltd, 2013. 152 p. ISBN 978-1-849-69873-3.

[Bakhshi 2017] BAKHSHI, T. State of the Art and Recent Research Advances in Software Defined Networking. *Wireless Communications and Mobile Computing*, Hindawi, v. 2017, p. 1–35, 2017. ISSN 1530-8669.

[Banse e Rangarajan 2015] BANSE, C.; RANGARAJAN, S. A Secure Northbound Interface for SDN Applications. In: *IEEE Trustcom/BigDataSE/ISPA*. Helsinki: IEEE, 2015. v. 1, p. 834–839.

[Bari *et al.* 2013] BARI, M. *et al.* PolicyCop: An Autonomic QoS Policy Enforcement Framework for Software-Defined Networks. In: *SDN for Future Networks and Services (SDN4FNS)*. Trento, Italy: IEEE, 2013. p. 1–7.

[Basili e Rombach 1988] BASILI, V. R.; ROMBACH, H. D. The TAME Project: Towards Improvement-oriented Software Environments. *IEEE Transactions on Software Engineering*, IEEE, v. 14, n. 6, p. 758–773, 1988.

[Belzarena *et al.* 2016] BELZARENA, P. *et al.* SDN-based Overlay Networks for QoS-aware Routing. In: *Proceedings of the 2016 Workshop on Fostering Latin-American Research in Data Communication Networks*. New York, NY, USA: ACM, 2016. (LANCOMM '16), p. 19–21. ISBN 978-1-4503-4426-5. Disponível em: <http://doi.acm.org/10.1145/2940116.2940121>.

[Berde *et al.* 2014] BERDE, P. *et al.* ONOS: Towards an Open, Distributed SDN OS. In: *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*. New York, NY, USA: ACM, 2014. (HotSDN '14), p. 1–6. ISBN 978-1-4503-2989-7.

[Berman *et al.* 2014] BERMAN, M. *et al.* GENI: A Federated Testbed for Innovative Network Experiments. *Computer Networks*, v. 61, n. 14, p. 5–23, 2014. ISSN 1389-1286. Special Issue on Future Internet Testbeds. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1389128613004507>.

[Big Switch Networks 2013] Big Switch Networks. *Project Floodlight*. 2013. Disponível em: <http://www.projectfloodlight.org/>.

[Big Switch Networks 2016] Big Switch Networks. *Big Switch Networks, Inc.* 2016. Disponível em: <http://www.bigswitch.com>.

[Blial, Mamoun e Benaini 2016] BLIAL, O.; MAMOUN, M. B.; BENAINI, R. An Overview on SDN Architectures with Multiple Controllers. *Journal of Computer Networks and Communications*, Hindawi Publishing Corporation, v. 2016, n. 9396525, p. 1–8, 2016. Disponível em: <http://dx.doi.org/10.1155/2016/9396525>.

[Bollapragada, Murphy e White 2000] BOLLAPRAGADA, V.; MURPHY, C.; WHITE, R. *Inside Cisco IOS Software Architecture*. 1st. ed. Indiana, USA: Cisco Press, 2000. 240 p. ISBN 978-1-587-05816-5.

[Bondkovskii *et al.* 2016] BONDKOVSKII, A. *et al.* Qualitative Comparison of Open-source SDN Controllers. In: *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. Istanbul, Turkey: IEEE, 2016. p. 889–894.

[Boucadair e Jacquenet 2014] BOUCADAIR, M.; JACQUENET, C. *Software-Defined Networking: A Perspective from within a Service Provider Environment*. IETF, 2014. Internet Draft. (Request for Comments, 7149). Disponível em: <https://tools.ietf.org/html/rfc7149>.

[Bray 2014] BRAY, T. *The JavaScript Object Notation (JSON) Data Interchange Format*. IETF, 2014. RFC 7159 (Proposed Standard). (Request for Comments, 7159). Disponível em: <https://tools.ietf.org/rfc/rfc7159.txt>.

[Burke 2013] BURKE, B. *Restful Java with Jax-RS 2.0*. 2nd. ed. CA, USA: O'Reilly Media, Inc., 2013. 392 p. ISBN 978-1-449-36134-1.

[Caba e Soler 2015] CABA, C.; SOLER, J. APIs for QoS Configuration in Software Defined Networks. In: *Proceedings of the 1st IEEE Conference on Network Softwarization (NetSoft 2015)*. London, UK: IEEE, 2015. p. 1–5.

[Caesar *et al.* 2005] CAESAR, M. *et al.* Design and Implementation of a Routing Control Platform. In: *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation*. Berkeley, CA, USA: USENIX Association, 2005. (NSDI'05, v. 2), p. 15–28.

[Cai, Cox e Ng 2011] CAI, Z.; COX, A. L.; NG, T. S. E. *Maestro: A System for Scalable OpenFlow Control*. Departament of Computer Science, Rice University, 2011. Disponível em: <http://www.cs.rice.edu/~eugeneng/papers/TR10-11.pdf>.

[Calvert *et al.* 1998] CALVERT, K. *et al.* Directions in Active Networks. *Communications Magazine, IEEE*, v. 36, n. 10, p. 72–78, October 1998. ISSN 0163-6804.

[Casado, Foster e Guha 2014] CASADO, M.; FOSTER, N.; GUHA, A. Abstractions for Software-defined Networks. *Communications ACM*, ACM, New York, NY, USA, v. 57, n. 10, p. 86–95, September 2014. ISSN 0001-0782.

[Casey, Sutton e Sprintson 2014] CASEY, C. J.; SUTTON, A.; SPRINTSON, A. tinyNBI: Distilling an API from Essential OpenFlow Abstractions. In: *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*. New York, NY, USA: ACM, 2014. (HotSDN '14), p. 37–42. ISBN 978-1-4503-2989-7.

[Chen e Wu 2017] CHEN, X.; WU, T. Towards the Semantic Web Based Northbound Interface for SDN Resource Management. In: *11th International Conference on Semantic Computing (ICSC 2017)*. San Diego, CA, USA: IEEE, 2017. p. 40–47.

[Chown *et al.* 2014] CHOWN, T. *et al.* OFERTIE: Enabling SDN/Openflow Support in Real-Time Online Interactive Applications. In: *Preceedings of TERENA Networking Conference (TNC2014)*. Dublin, Ireland: [s.n.], 2014. p. 1–5. Disponível em: <https://tnc2014.terena.org/getfile/1016>.

[Cook, Campbell e Day 1979] COOK, T. D.; CAMPBELL, D. T.; DAY, A. *Quasi-Experimentation: Design & Analysis Issues for Field Settings*. Boston, USA: Houghton Mifflin, 1979. v. 351.

[Daradkeh *et al.* 2016] DARADKEH, Y. I. *et al.* On Standards for Application Level Interfaces in SDN. *International Journal of Advanced Computer Science and Applications, IJACSA*, v. 7, n. 10, p. 45–51, 2016. Disponível em: <http://dx.doi.org/10.14569/IJACSA.2016.071006>.

[DLUX 2017] DLUX. *OpenDaylight User Interface (DLUX)*. 2017. Disponível em: <http://docs.opendaylight.org/en/stable-carbon/getting-started-guide/common-features/dlux.html>.

[Doriguzzi-Corin *et al.* 2016] DORIGUZZI-CORIN, R. *et al.* Reusability of Software-defined Networking Applications: A Runtime, Multi-controller Approach. In: *12th International Conference on Network and Service Management (CNSM)*. Montreal, QC, Canada: IEEE, 2016. p. 209–215.

[Duckett 2013] DUCKETT, C. *Software Defined Networking: HP has an App Store for that*. ZDNet, 2013. Disponível em: <http://www.zdnet.com/software-defined-networking-hp-has-an-app-store-for-that-7000021365/>.

[Durbin e Watson 1950] DURBIN, J.; WATSON, G. S. Testing for Serial Correlation in Least Squares Regression: I. *Biometrika*, JSTOR, v. 37, n. 3/4, p. 409–428, 1950.

[Erickson 2013] ERICKSON, D. The Beacon OpenFlow controller. In: *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. New York, NY, USA: ACM, 2013. (HotSDN '13), p. 13–18. ISBN 978-1-4503-2178-5.

[Facebook Inc. 2016] Facebook Inc. *Facebook Graph API*. 2016. Disponível em: <https://developers.facebook.com/docs/graph-api>.

[Feamster, Rexford e Zegura 2014] FEAMSTER, N.; REXFORD, J.; ZEGURA, E. The Road to SDN: An Intellectual History of Programmable Networks. *ACM SIGCOMM Computer Communication Review*, ACM, v. 44, n. 2, p. 87–98, 2014.

[Ferguson *et al.* 2013] FERGUSON, A. D. *et al.* Participatory Networking: An API for Application Control of SDNs. *SIGCOMM Computer Communications Review*, ACM, New York, NY, USA, v. 43, n. 4, p. 327–338, August 2013. ISSN 0146-4833.

[Field, Miles e Field 2012] FIELD, A. P.; MILES, J.; FIELD, Z. *Discovering Statistics using R*. London: SAGE, 2012.

[Fielding *et al.* 1999] FIELDING, R. *et al. Hypertext Transfer Protocol–HTTP/1.1*. IETF, 1999. RFC 2616 (Draft Standard). (Request for Comments, 2616). Disponível em: <https://tools.ietf.org/rfc/rfc2616.txt>.

[Fielding 2000] FIELDING, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. Tese (Doutorado) — University of California, Irvine, 2000. Disponível em: <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>.

[Foster *et al.* 2013] FOSTER, N. *et al.* Languages for Software-defined Networks. *Communications Magazine, IEEE*, v. 51, n. 2, p. 128–134, 2013. ISSN 0163-6804.

[Gamma *et al.* 1995] GAMMA, E. *et al. Design Patterns: Elements of Reusable Object-oriented Software*. Boston: Addison-Wesley, 1995.

[Gartner Inc. 2014] Gartner Inc. *HP's SDN App Store Needs Open Platform to Change the Future of Networking*. 2014. Disponível em: <https://www.gartner.com/doc/2859722/hps-sdn-app-store-needs>.

[Ghodsi *et al.* 2011] GHODSI, A. *et al.* Intelligent Design Enables Architectural Evolution. In: *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*. New York, NY, USA: ACM, 2011. (HotNets-X), p. 3:1–3:6. ISBN 978-1-4503-1059-8.

[Google Inc. 2016] Google Inc. *Google+ REST API*. 2016. Disponível em: <https://developers.google.com/+/web/api/rest/#api>.

[Google, Inc. 2017] Google, Inc. *YouTube: Definições de Codificação para Carregamento Recomendadas*. 2017. Disponível em: <https://support.google.com/youtube/answer/1722171>.

[Gorlatch e Humernbrum 2015] GORLATCH, S.; HUMERNBRUM, T. Enabling High-level QoS Metrics for Interactive Online Applications using SDN. In: IEEE. *nternational Conference on Computing, Networking and Communications (ICNC)*. Garden Grove, CA: IEEE, 2015. p. 707–711.

[Gorlatch, Humernbrum e Glinka 2014] GORLATCH, S.; HUMERNBRUM, T.; GLINKA, F. Improving QoS in Real-time Internet Applications: from Best-effort to Software-Defined Networks. In: *International Conference on Computing, Networking and Communications (ICNC)*. Honolulu, Hawaii, USA: IEEE, 2014. p. 189–193.

[Greene 2009] GREENE, K. *MIT Tech Review 10 Breakthrough Technologies: Software-defined Networking*. 2009. Disponível em: <http://www2.technologyreview.com/article/412194/tr10-software-defined-networking/>.

[Gude *et al.* 2008] GUDE, N. *et al.* NOX: Towards an Operating System for Networks. *ACM SIGCOMM Computer Communication Review*, ACM, New York, NY, USA, v. 38, July 2008. ISSN 0146-4833.

[Haleplidis *et al.* 2015] HALEPLIDIS, E. *et al. Software-Defined Networking (SDN): Layers and Architecture Terminology*. IETF, 2015. RFC 7426 (Informational). Disponível em: <https://tools.ietf.org/rfc/rfc7426.txt>.

[Hawley 2013] HAWLEY, D. *SDN's Northbound API*. 2013. Disponível em: <https://www.sdxcentral.com/articles/featured/sdnjourney-sdn-northbound-api-metzler/2013/09/>.

[Hewlett-Packard 2013a] Hewlett-Packard. *SDN App Store*. 2013a. Disponível em: <https://saas.hpe.com/marketplace/sdn>.

[Hewlett-Packard 2013b] Hewlett-Packard. *HP SDN Controller Architecture*. 2013b. 14 p. Disponível em: <http://h17007.www1.hp.com/docs/networking/solutions/sdn/devcenter/06_-_HP_SDN_Controller_Architecture_TSG_v1_3013-10-01.pdf>.

[Hewlett-Packard 2014] Hewlett-Packard. *HP Open SDN Ecosystem and SDN Applications*. 2014. 4 p. Disponível em: <http://h17007.www1.hp.com/docs/sdn/4AA5-1871ENW.pdf>.

[Hewlett-Packard 2016] Hewlett-Packard. *HP Virtual Application Networks (VAN) SDN Controller*. 2016. Disponível em: <http://h20195.www2.hp.com/v2/GetDocument.aspx?docname=4AA4-9827ENW&doctype=data%20sheet&doclang=EN_US&searchquery=&cc=us&lc=en>.

[Hewlett-Packard 2016b] Hewlett-Packard. *HP SDN Developer Kit (SDK)*. 2016b. Disponível em: <http://h17007.www1.hp.com/ie/en/networking/solutions/technology/sdn/devcenter/index.aspx#tab=TAB2>.

[Hewlett-Packard 2017] Hewlett-Packard. *HPE FlexNetwork 5130 24G 4SFP+ EI Brazil Switch Series (JG975A) - Documentação de Produto*. 2017. Disponível em: <https://www.hpe.com/h20195/v2/default.aspx?cc=br&lc=pt&oid=6908630>.

[Hu, Hao e Bao 2014] HU, F.; HAO, Q.; BAO, K. A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation. *IEEE Communications Surveys Tutorials*, v. 16, n. 4, p. 2181–2206, May 2014. ISSN 1553-877X.

[Huang e Griffioen 2013] HUANG, S.; GRIFFIOEN, J. HyperNet Games: Leveraging SDN Networks to Improve Multiplayer Online Games. In: *Computer Games: AI, Animation, Mobile, Interactive Multimedia, Educational Serious Games (CGAMES), 2013 18th International Conference on*. Louisville, KY: IEEE, 2013. p. 74–78. ISBN 978-1-4799-0818-9.

[Humernbrum *et al.* 2015] HUMERNBRUM, T. *et al.* RTF+Shark: Using Software-Defined Networks for Multiplayer Online Games. In: IEEE. *14th IEEE/ACM International Workshop on Network and Systems Support for Games (NetGames)*. Zagreb: IEEE, 2015.

[Humernbrum, Glinka e Gorlatch 2014] HUMERNBRUM, T.; GLINKA, F.; GORLATCH, S. Using Software-Defined Networking for Real-Time Internet Applications. In: *Proceedings of the International MultiConference of Engineers and Computer Scientists (IMECS)*. Hong Kong, China: IAENG, 2014. v. 1, p. 150–155.

[Humernbrum, Glinka e Gorlatch 2014a] HUMERNBRUM, T.; GLINKA, F.; GORLATCH, S. A Northbound API for QoS Management in Real-Time Interactive Applications on Software-Defined Networks. *Journal of Communications*, v. 9, n. 8, p. 607–615, August 2014a.

[IANA 2017] IANA. *Internet Assigned Numbers Authority - Protocol Numbers*. 2017. Disponível em: <https://www.iana.org/assignments/protocol-numbers/protocol-numbers.txt>.

[IANA 2017a] IANA. *Internet Assigned Numbers Authority - IEEE 802 Numbers*. 2017a. Disponível em: <https://www.iana.org/assignments/ieee-802-numbers/ieee-802-numbers.txt>.

[Infosim 2015] Infosim. *StableNet - Unified Network and Service Management Suite*. 2015. Disponível em: <http://www.infosim.net/stablenet/>.

[Instagram Inc. 2016] Instagram Inc. *Instagram API Platform*. 2016. Disponível em: <https://www.instagram.com/developer>.

[Iperf 2008] Iperf. 2008. Disponível em: <http://iperf.sourceforge.net>.

[Ishimori *et al.* 2013] ISHIMORI, A. *et al.* Control of Multiple Packet Schedulers for Improving QoS on OpenFlow/SDN Networking. In: *Second European Workshop on Software Defined Networks (EWSDN/2013)*. Berlin, Germany: IEEE, 2013. p. 81–86. ISSN 2379-0350.

[Izard 2017] IZARD, R. *Static Entry Pusher API*. 2017. Disponível em: <https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343518/Static+Entry+Pusher+API>.

[Izard 2017a] IZARD, R. *Floodlight Web GUI*. 2017a. Disponível em: <https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/40403023/Web+GUI>.

[Jain 1991] JAIN, R. The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling. John Wiley & Sons, 1991.

[Jain *et al.* 2013] JAIN, S. *et al.* B4: Experience with a Globally-deployed Software Defined WAN. In: *Proceedings of the ACM SIGCOMM 2013*. New York, NY, USA: ACM, 2013. (SIGCOMM '13), p. 3–14. ISBN 978-1-4503-2056-6.

[Jakobsson, Mulligan e Unmehopa 2012] JAKOBSSON, S.; MULLIGAN, C.; UNMEHOPA, M. Research and Reality: The Evolution of Open Network API Standards. In: IEEE. *International Conference on Communications (ICC)*. Ottawa, ON: IEEE, 2012. p. 6901–6905. ISSN 1550-3607.

[Juristo e Moreno 2013] JURISTO, N.; MORENO, A. M. *Basics of Software Engineering Experimentation*. New York, NY: Springer Science & Business Media, 2013.

[Kalebaila 2015] KALEBAILA, G. *Making SDN Real for Enterprises*. 2015. Disponível em: <http://www.cisco.com/c/dam/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/idc-making-sdn-real-for-enterprises.pdf>.

[Karakus e Durresi 2017] KARAKUS, M.; DURRESI, A. Quality of Service (QoS) in Software Defined Networking (SDN): A Survey. *Journal of Network and Computer Applications*, v. 80, p. 200–218, 2017. ISSN 1084-8045. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1084804516303186>.

[Khattak, Awais e Iqbal 2014] KHATTAK, Z. K.; AWAIS, M.; IQBAL, A. Performance Evaluation of OpenDaylight SDN Controller. In: *Parallel and Distributed Systems (ICPADS), 2014 20th IEEE International Conference on*. Hsinchu, Taiwan: IEEE, 2014. p. 671–676. ISSN 1521-9097.

[Khondoker *et al.* 2014] KHONDOKER, R. *et al.* Feature-based Comparison and Selection of Software-Defined Networking (SDN) Controllers. In: *World Congress on Computer Applications and Information Systems (WCCAIS)*. Hammamet: IEEE, 2014. p. 1–7.

[Kim e Feamster 2013] KIM, H.; FEAMSTER, N. Improving Network Management with Software-Defined Networking. *Communications Magazine, IEEE*, v. 51, n. 2, p. 114–119, 2013. ISSN 0163-6804.

[Kondwilkar *et al.* 2015] KONDWILKAR, A. *et al.* Can an SDN-based Network Management System use Northbound REST APIs to Communicate Network Changes to the Application Layer? *Interdisciplinary Telecom Program, Capstone Research Project, University of Colorado Boulder*, p. 1–10, 2015.

[Kreutz *et al.* 2015] KREUTZ, D. *et al.* Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*, IEEE, v. 103, n. 1, p. 14–76, 2015. Disponível em: <http://arxiv.org/abs/1406.0440>.

[Kumar e Sood 2017] KUMAR, D.; SOOD, M. Software Defined Networks (SDN): Approaches Needed for Upgradation of SDNs. *International Journal of Engineering Sciences and Research Technology (IJESRT)*, v. 6, n. 2, p. 521–526, 2017.

[Lantz, Heller e McKeown 2010] LANTZ, B.; HELLER, B.; MCKEOWN, N. A Network in a Laptop: Rapid Prototyping for Software-defined Networks. In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. New York, NY, USA: ACM, 2010. (Hotnets-IX), p. 19:1–19:6. ISBN 978-1-4503-0409-2.

[Layeghy, Pakzad e Portmann 2016] LAYEGHY, S.; PAKZAD, F.; PORTMANN, M. SCOR: Software-defined Constrained Optimal Routing Platform for SDN. *CoRR*, abs/1607.03243, 2016. Disponível em: <http://arxiv.org/abs/1607.03243>.

[Layeghy, Pakzad e Portmann 2017] LAYEGHY, S.; PAKZAD, F.; PORTMANN, M. A New QoS Routing Northbound Interface for SDN. *Australian Journal of Telecommunications and the Digital Economy*, Telecommunication Society of Australia, Melbourne, VIC, Australia, v. 5, n. 1, p. 92, Mar 2017. ISSN 2203-1693. Disponível em: <https://telsoc.org/ajtde/index.php/ajtde/article/view/91>.

[Levene 1960] LEVENE, H. Contributions to Probability and Statistics. *Essays in Honor of Harold Hotelling*, Stanford University Press, p. 278–292, 1960.

[Li e Chou 2011] LI, L.; CHOU, W. Design and Describe REST API without Violating REST: A Petri Net-based Approach. In: IEEE. *International Conference on Web Services (ICWS)*. Washington DC: IEEE, 2011. p. 508–515.

[Lin *et al.* 2014] LIN, T. *et al.* Enabling SDN Applications on Software-Defined Infrastructure. In: *Network Operations and Management Symposium (NOMS)*. Krakow, Poland: IEEE, 2014. p. 1–7. ISSN 1542-1201.

[Lopes *et al.* 2016] LOPES, F. A. *et al.* A Software Engineering Perspective on SDN Programmability. *IEEE Communications Surveys Tutorials*, v. 18, n. 2, p. 1255–1272, Secondquarter 2016. ISSN 1553-877X.

[Luo *et al.* 2012] LUO, M. *et al.* Sox–A Generalized and Extensible Smart Network OpenFlow Controller. In: *Proceedings of the First SDN World Congress*. Damsdadt, Germany: [s.n.], 2012.

[Macedonia e Brutzman 1994] MACEDONIA, M.; BRUTZMAN, D. MBone Provides Audio and Video Across the Internet. *Computer*, v. 27, n. 4, p. 30–36, 1994. ISSN 0018-9162.

[Marschke, Doyle e Moyer 2015] MARSCHKE, D.; DOYLE, J.; MOYER, P. *Software Defined Networking (SDN): Anatomy of OpenFlow Volume I*. North Carolina, USA: Lulu Publishing Services, 2015. v. 1.

[Masoudi e Ghaffari 2016] MASOUDI, R.; GHAFFARI, A. Software Defined Networks: A Survey. *Journal of Network and Computer Applications*, Elsevier, New York, NY, USA, v. 67, p. 1–25, 2016.

[McCauley 2012] MCCAULEY, M. *POX: A Python-based Openflow Controller*. 2012. Disponível em: <https://github.com/noxrepo/pox>.

[McKeown *et al.* 2008] MCKEOWN, N. *et al.* OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Computer Communication Review*, ACM, New York, NY, USA, v. 38, n. 2, p. 69–74, March 2008. ISSN 0146-4833.

[Megyesi *et al.* 2017] MEGYESI, P. *et al.* Challenges and Solution for Measuring Available Bandwidth in Software Defined Networks. *Computer Communications*, v. 99, n. Supplement C, p. 48–61, 2017. ISSN 0140-3664. Disponível em: <http://www.sciencedirect.com/science/article/pii/S014036641630648X>.

[Merwe *et al.* 1998] MERWE, J. Van der *et al.* The Tempest: a Practical Framework for Network Programmability. *Network, IEEE*, v. 12, n. 3, p. 20–28, May 1998. ISSN 0890-8044.

[Metzler 2014] METZLER, J. *Where Do We Stand with SDN's Northbound Interface?* 2014. Disponível em: <http://www.webtorials.com/content/2014/04/where-do-we-stand-with-sdns-northbound-interface.html>.

[MUL 2014] MUL. *Open MUL - High Performance SDN*. 2014. Disponível em: <http://www.openmul.org/>.

[Neto *et al.* 2017] NETO, E. *et al.* MISSIn: Orquestração Interativa de Infraestruturas SDN. In: *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2017)*. Belém, Brasil: [s.n.], 2017. Disponível em: <https://sbrc2017.ufpa.br/salao-ferramentas/artigos-aceitos/>.

[Neutron 2012] NEUTRON. *Neutron's Developer Documentation*. 2012. Disponível em: <http://docs.openstack.org/developer/neutron>.

[Nichols *et al.* 1998] NICHOLS, K. *et al. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*. IETF, 1998. RFC 2474 (Proposed Standard). (Request for Comments, 2474). Disponível em: <https://tools.ietf.org/rfc/rfc2474.txt>.

[Nippon Telegraph and Telephone Corporation 2012] Nippon Telegraph and Telephone Corporation. *Ryu Network Operating System*. 2012. Disponível em: <http://osrg.github.com/ryu/>.

[Northbound Networks 2017] Northbound Networks. *Zodiac FX - SDN Development Board*. 2017. Disponível em: <https://northboundnetworks.com/products/zodiac-fx>.

[Nunes *et al.* 2014] NUNES, B. *et al.* A Survey of Software-defined Networking: Past, Present, and Future of Programmable Networks. *Communications Surveys Tutorials, IEEE*, v. 16, n. 3, p. 1617–1634, Third 2014. ISSN 1553-877X.

[ONF 2011] ONF. *Open Networking Foundation*. 2011. Disponível em: <https://www.opennetworking.org/>.

[ONF 2012] ONF. *OpenFlow Switch Specification (Version 1.3.0 - Wire Protocol 0x04)*. 2012. Disponível em: <https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf>.

[ONF 2013] ONF. *Charter: Open Networking Foundation Northbound Interface Working Group*. 2013. Disponível em: <https://www.opennetworking.org/images/stories/downloads/working-groups/charter-nbi.pdf>.

[ONF 2014] ONF. *SDN Architecture - ONF TR-502*. 2014. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN_ARCH_1.0_06062014.pdf>.

[ONF 2014a] ONF. *OpenFlow Management and Configuration Protocol (OF-CONFIG) v1.2*. 2014a. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-config/of-config-1.2.pdf>.

[ONF 2016] ONF. *SDN Architecture - ONF TR-521*. 2016. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR-521_SDN_Architecture_issue_1.1.pdf>.

[OpenDaylight 2013] OpenDaylight. *OpenDaylight: A Linux Foundation Collaborative Project*. 2013. Disponível em: <http://www.opendaylight.org>.

[OpenDaylight 2017] OpenDaylight. *Installing OpenDaylight*. 2017. Disponível em: <http://docs.opendaylight.org/en/stable-carbon/getting-started-guide/installing_opendaylight.html>.

[OpenDaylight 2017a] OpenDaylight. *OpenDaylight OpenFlow Plugin:End to End Flows*. 2017a. Disponível em: <https://wiki.opendaylight.org/view/OpenDaylight_OpenFlow_Plugin:End_to_End_Flows>.

[OpenDaylight 2017b] OpenDaylight. *OpenDaylight Controller – MD-SAL Explained*. 2017b. Disponível em: <https://wiki.opendaylight.org/view/OpenDaylight_Controller:MD-SAL:Explained>.

[OpenStack 2014] OpenStack. *The OpenStack Cloud Software Project*. 2014. Disponível em: <http://openstack.org>.

[Palma *et al.* 2014] PALMA, D. *et al.* The QueuePusher: Enabling Queue Management in OpenFlow. In: *Third European Workshop on Software Defined Networks*. [S.l.: s.n.], 2014. p. 125–126. ISSN 2379-0350.

[Pan *et al.* 2017] PAN, T. *et al.* OpenSched: Programmable Packet Queuing and Scheduling for Centralized QoS Control. In: *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS 2017)*. Beijing, China: IEEE, 2017. p. 95–96.

[Pfaff e Davie 2013] PFAFF, B.; DAVIE, B. *The Open vSwitch Database Management Protocol*. IETF, 2013. RFC 7047 (Informational). (Request for Comments, 7047). Disponível em: <http://www.ietf.org/rfc/rfc7047.txt>.

[Pfaff *et al.* 2009] PFAFF, B. *et al.* Extending Networking into the Virtualization Layer. In: *Proceedings of Workshop on Hot Topics in Networks (HotNets-VIII)*. New York, NY: ACM, 2009.

[Pham e Hoang 2016] PHAM, M.; HOANG, D. B. SDN Applications - The Intent-based Northbound Interface Realisation for Extended Applications. In: *IEEE NetSoft Conference and Workshops (NetSoft 2016)*. Seoul, South Korea: IEEE, 2016. p. 372–377.

[Pitt 2013] PITT, D. *Compass Check: ONF & the Northbound API*. 2013. Disponível em: <http://www.lightreading.com/carrier-sdn/sdn-architectures/compass-check-onf-and-the-northbound-api(s)/a/d-id/706475>.

[Postman 2017] POSTMAN. *Postman: a Complete API Development Environment*. 2017. Disponível em: <https://www.getpostman.com/postman>.

[Reitblatt *et al.* 2012] REITBLATT, M. *et al.* Abstractions for Network Update. In: *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures and Protocols for Computer Communication*. New York, NY, USA: ACM, 2012. (SIGCOMM '12), p. 323–334. ISBN 978-1-4503-1419-0.

[Richardson, Amundsen e Ruby 2013] RICHARDSON, L.; AMUNDSEN, M.; RUBY, S. *RESTful Web APIs*. 1st. ed. CA, USA: O'Reilly Media, Inc., 2013. 404 p. ISBN 978-1-449-35608-8.

[Richardson e Ruby 2008] RICHARDSON, L.; RUBY, S. *RESTful Web Services*. 1st. ed. CA, USA: O'Reilly Media, Inc., 2008. 454 p. ISBN 978-0-596-52926-0.

[Rivera, Fei e Griffioen 2016] RIVERA, S.; FEI, Z.; GRIFFIOEN, J. RAPTOR: A REST API translaTOR for OpenFlow Controllers. In: *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. San Francisco, CA, USA: IEEE, 2016. p. 328–333.

[Rotsos *et al.* 2017] ROTSOS, C. *et al.* Network Service Orchestration Standardization: A Technology Survey. *Computer Standards & Interfaces*, Elsevier North-Holland, Inc., v. 54, Part 4, p. 203–215, 2017. ISSN 0920-5489. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0920548916302458>.

[Russinovich 2015] RUSSINOVICH, M. *Report from Open Networking Summit: Achieving Hyper-Scale with Software Defined Networking*. Microsoft Azure Blog, 2015. Disponível em: <https://azure.microsoft.com/pt-br/blog/report-from-open-networking-summit-achieving-hyper-scale-with-software-defined-networking>.

[Ryu Development Team 2017] Ryu Development Team. *ryu Documentation - Release 4.19*. 2017. Disponível em: <https://media.readthedocs.org/pdf/ryu/latest/ryu.pdf>.

[Ryubook 2016] RYUBOOK. *Ryubook Documentation*. 2016. Disponível em: <https://osrg.github.io/ryu-book/en/html/rest_qos.html>.

[Sahoo *et al.* 2017] SAHOO, K. S. *et al.* Software Defined Network: The Next Generation Internet Technology. *International Journal of Wireless and Microwave Technologies*, Modern Education and Computer Science Press, v. 7, p. 13–24, 2017. Disponível em: <http://dspace.nitrkl.ac.in/dspace/handle/2080/2685>.

[Salman *et al.* 2016] SALMAN, O. *et al.* SDN Controllers: A Comparative Study. In: *18th Mediterranean Electrotechnical Conference (MELECON 2016)*. Lemesos, Cyprus: IEEE, 2016. p. 1–6.

[Schneider *et al.* 2014] SCHNEIDER, F. *et al.* Standardizations of SDN and its Practical Implementation. *NEC Technical Journal, Special Issue on SDN and Its Impact on Advanced ICT Systems*, v. 8, n. 2, 2014.

[Shapiro e Wilk 1965] SHAPIRO, S. S.; WILK, M. B. An Analysis of Variance Test for Normality (complete samples). *Biometrika*, JSTOR, v. 52, n. 3/4, p. 591–611, 1965.

[Shin, Nam e Kim 2012] SHIN, M.-K.; NAM, K.-H.; KIM, H.-J. Software-defined Networking (SDN) A Reference Architecture and Open APIs. In: IEEE. *International Conference on ICT Convergence (ICTC)*. Jeju Island: IEEE, 2012. p. 360–361.

[Shin *et al.* 2014] SHIN, S. *et al.* Rosemary: A Robust, Secure, and High-performance Network Operating System. In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA: ACM, 2014. (CCS '14), p. 78–89. ISBN 978-1-4503-2957-6. Disponível em: <http://doi.acm.org/10.1145/2660267.2660353>.

[Sieber *et al.* 2016] SIEBER, C. *et al.* Towards a Programmable Management Plane for SDN and Legacy Networks. In: *IEEE NetSoft Conference and Workshops (NetSoft 2016)*. Seoul, South Korea: IEEE, 2016. p. 319–327.

[Sieber *et al.* 2015] SIEBER, C. *et al.* Network Configuration with Quality of Service Abstractions for SDN and Legacy Networks. In: *IFIP/IEEE International Symposium on Integrated Network Management (IM 2015)*. Ottawa, Canada: IEEE, 2015. p. 1135–1136. ISSN 1573-0077.

[Singh e Jha 2017] SINGH, S.; JHA, R. K. A Survey on Software Defined Networking: Architecture for Next Generation Network. *Journal of Network and Systems Management*, v. 25, n. 2, p. 321–374, Apr 2017. ISSN 1573-7705. Disponível em: <https://doi.org/10.1007/s10922-016-9393-9>.

[Solingen *et al.* 2002] SOLINGEN, R. V. *et al.* Goal Question Metric (GQM) Approach. *Encyclopedia of Software Engineering*, Wiley Online Library, 2002.

[Song 2013] SONG, H. Protocol-oblivious Forwarding: Unleash the Power of SDN Through a Future-proof Forwarding Plane. In: *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*. New York, NY, USA: ACM, 2013. (HotSDN '13), p. 127–132. ISBN 978-1-4503-2178-5.

[Spinor-GmbH 2015] Spinor-GmbH. *Shark 3D-Overview & Features*. 2015. Disponível em: <http://www.spinor.com/features.html>.

[Suh *et al.* 2016] SUH, D. *et al.* On Performance of OpenDaylight Clustering. In: *IEEE NetSoft Conference and Workshops (NetSoft)*. Seoul, South Korea: IEEE, 2016. p. 407–410.

[Takamiya e Karanatsios 2012] TAKAMIYA, Y.; KARANATSIOS, N. *Trema OpenFlow Controller Framework*. 2012. Disponível em: <https://github.com/trema/trema>.

[Tennenhouse *et al.* 1997] TENNENHOUSE, D. *et al.* A Survey of Active Network Research. *Communications Magazine, IEEE*, v. 35, n. 1, p. 80–86, 1997. ISSN 0163-6804.

[Tijare 2016] TIJARE, D. V. P. V. The Northbound APIs of Software Defined Networks. *International Journal of Engineering Sciences & Research Technology*, v. 5, n. 10, p. 501–513, October 2016. Disponível em: <https://doi.org/10.5281/zenodo.160891>.

[Trois *et al.* 2016] TROIS, C. *et al.* A Survey on SDN Programming Languages: Toward a Taxonomy. *IEEE Communications Surveys Tutorials*, v. 18, n. 4, p. 2687–2712, April 2016. ISSN 1553-877X.

[Twitter Inc. 2016] Twitter Inc. *Twitter REST APIs*. 2016. Disponível em: <https://dev.twitter.com/rest/public>.

[Valluvan, Manoranjitham e Nagarajan 2016] VALLUVAN, S.; MANORANJITHAM, T.; NAGARAJAN, V. A Study of SDN Controllers. *International Journal of Pharmacy & Technology (IJPT)*, v. 8, n. 4, p. 5235–5242, December 2016. ISSN 0975-766X. Disponível em: <http://www.ijptonline.com/wp-content/uploads/2017/01/5234-5242.pdf>.

[Vasconcelos *et al.* 2014] VASCONCELOS, R. C. *et al.* Evaluating the Geographic Distance Effects in Communication Between Hosts and a Controller on a Software-defined Network. In: *Proceedings of the 13th International Conference WWW/INTERNET 2014 (ICWI)*. Porto, Portugal: IADIS, 2014. p. 275–282. ISBN 978-989-8533-24-1.

[Vasconcelos *et al.* 2015] VASCONCELOS, R. C. *et al.* Towards an SDN-App Store: A Generic Northbound API for Software-Defined Networks. In: *Proceedings of the 14th International Conference WWW/INTERNET 2015 (ICWI)*. Maynooth, Dublin, Ireland: IADIS, 2015. p. 173–176. ISBN 978-989-8533-44-9.

[Vasconcelos *et al.* 2017] VASCONCELOS, R. C. *et al.* Enabling High-level Network Programming: A Northbound API for Software-Defined Networks. In: *2017 International Conference on Information Networking (ICOIN)*. Da Nang, Vietnam: IEEE, 2017. p. 662–667.

[W3C 2007] W3C. *SOAP Version 1.2: Messaging Framework (Second Edition)*. 2007. Disponível em: <http://www.w3.org/TR/soap12/>.

[W3C 2013] W3C. *SPARQL 1.1 Overview*. 2013. Disponível em: <https://www.w3.org/TR/sparql11-overview/>.

[Wallner 2015] WALLNER, R. *How to implement Quality Of Service using Floodlight*. 2015. Disponível em: <https://floodlight.atlassian.net/wiki/display/floodlightcontroller/How+to+implement+Quality+Of+Service+using+Floodlight>.

[Wallner e Cannistra 2013] WALLNER, R.; CANNISTRA, R. An SDN Approach: Quality of Service Using Big Switch's Floodlight Open-source Controller. *Proceedings of the Asia-Pacific Advanced Network*, v. 35, p. 14–19, 2013.

[Wohlin *et al.* 2012] WOHLIN, C. *et al. Experimentation in Software Engineering*. New York, NY: Springer Science & Business Media, 2012.

[Xia *et al.* 2015] XIA, W. *et al.* A Survey on Software-Defined Networking. *Communications Surveys & Tutorials*, IEEE, v. 17, n. 1, p. 27–51, 2015.

[Xie *et al.* 2015] XIE, J. *et al.* Control Plane of Software Defined Networks: A Survey. *Computer Communications*, v. 67, n. Supplement C, p. 1 – 10, 2015. ISSN 0140-3664. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0140366415002200>.

[Xu, Chen e Qian 2015] XU, C.; CHEN, B.; QIAN, H. Quality of Service Guaranteed Resource Management Dynamically in Software Defined Network. *Journal of Communications*, v. 10, n. 11, p. 843–850, November 2015.

[Yamei, Qing e Qi 2016] YAMEI, F.; QING, L.; QI, H. Research and Comparative Analysis of Performance Test on SDN Controllers. In: *First International Conference on Computer Communication and the Internet (ICCCI)*. Wuhan, China: IEEE, 2016. p. 207–210.

[Yan *et al.* 2015] YAN, J. *et al.* HiQoS: An SDN-based Multipath QoS Solution. *China Communications*, IEEE, v. 12, n. 5, p. 123–133, May 2015. ISSN 1673-5447.

[Yang *et al.* 2004] YANG, L. *et al. Forwarding and Control Element Separation (ForCES) Framework*. IETF, 2004. RFC 3746 (Informational). (Request for Comments, 3746). Disponível em: <https://tools.ietf.org/rfc/rfc3746.txt>.

[Yiakoumis, Schulz-Zander e Zhu 2011] YIAKOUMIS, Y.; SCHULZ-ZANDER, J.; ZHU, J. *Pantou : OpenFlow 1.0 for OpenWRT*. 2011. Disponível em: <http://www.openflow.org/wk/index.php/OpenFlow_1.0_for_OpenWRT>.

[Yu, Wundsam e Raju 2014] YU, M.; WUNDSAM, A.; RAJU, M. NOSIX: A Lightweight Portability Layer for the SDN OS. *SIGCOMM Computer Communication Review*, ACM, New York, NY, USA, v. 44, n. 2, p. 28–35, Apr 2014. ISSN 0146-4833.

[Zhang *et al.* 2015] ZHANG, S. *et al.* Ryuo: Using High Level Northbound API for Control Messages in Software-Defined Network. In: *Network Operations and Management Symposium (APNOMS), 2015 17th Asia-Pacific*. Busan: IEEE, 2015. p. 115–120.

[Zhang *et al.* 2018] ZHANG, Y. *et al.* A Survey on Software Defined Networking with Multiple Controllers. *Journal of Network and Computer Applications*, Academic Press Ltd., London, UK, v. 103, n. C, p. 101–118, Feb 2018. ISSN 1084-8045. Disponível em: <https://doi.org/10.1016/j.jnca.2017.11.015>.

[Zhou, Li e Chou 2014] ZHOU, W.; LI, L.; CHOU, W. SDN Northbound REST API with Efficient Caches. In: IEEE. *International Conference on Web Services (ICWS)*. Anchorage, AK: IEEE, 2014. p. 257–264. ISBN 978-1-4799-5053-9.

[Zhou *et al.* 2014] ZHOU, W. *et al.* REST API Design Patterns for SDN Northbound API. In: *28th International Conference on Advanced Information Networking and Applications Workshops (WAINA), 2014*. Victoria, BC: IEEE, 2014. p. 358–365. ISBN 978-1-4799-2652-7.

Apêndice A

Execução das Funções

Neste apêndice, é levada a efeito uma discussão sobre a forma de utilização de cada uma das funções oferecidas pela abordagem NoBI para programar a rede SDN em alto nível.

Do ponto de vista de utilização, o gerente de redes deve apenas acessar o corpo de funcionalidades (URIs) da interface NoBI e enviar mensagens HTTP (do inglês *Hypertext Transfer Protocol*) para orquestrar o controlador SDN e instalar/desinstalar regras de controle na rede. A NoBI irá abstrair toda a complexidade, configurações e detalhes internos de operação do controlador, além de oferecer ao gerente URIs padronizadas e que podem ser reutilizadas nos controladores (*Floodlight* ou *OpenDaylight*) sem esforço. Para garantir a total portabilidade do corpo de funcionalidades NoBI, cada uma das *engines* (Capítulo 4) é capaz de traduzir, em tempo de execução, as invocações às URIs em instruções específicas do controlador-alvo existente. Este último, por sua vez, as repassará ao plano de dados SDN.

Conforme mencionado no Capítulo 4, o arcabouço da interface *northbound* proposta nesta tese se encontra integrado ao *Floodlight* versão 1.2 [Big Switch Networks 2013] e *OpenDaylight Beryllium-SR4* [OpenDaylight 2013]. Em particular, cada uma das funções (URIs) apresentadas na Tabela 4.3 do Capítulo 4 são acessadas através da porta 8080 do controlador existente. Sendo assim, um exemplo válido de chamada (invocação) de função seria: **`http://<controller-ip>:8080/nobi/controller/topology`** [1]. Há outras funções, entretanto, que necessitam de receber argumentos, para que a comunicação com o controlador e a consequente programação da rede ocorram corretamente.

Além de acessar algumas das URIs diretamente, o gerente também pode invocá-las por

[1] Neste exemplo, o termo **`<controller-ip>`** representa o endereço IP atribuído ao controlador SDN.

meio da implementação de um cliente HTTP simples. Hoje em dia, boa parte das linguagens de programação modernas já possuem bibliotecas de classes que permitem criar clientes HTTP muito rapidamente. Por exemplo, no caso da linguagem *Java*, pode-se usar a classe `HttpURLConnection`[2] do pacote `java.net` para esta finalidade. Alternativamente, pode-se utilizar um aplicativo cliente HTTP já pronto e gratuito [Postman 2017]. Sendo assim, a forma de acesso às URIs da abordagem proposta é flexível e pode ser mantida simples.

A.1 Corpo de Funções

A interface *northbound* NoBI dispõe dos seguintes serviços fundamentais:

- **`/nobi/controller/topology`:** a presente função permite ao gerente de redes comunicar-se com o controlador (*Floodlight* ou *OpenDaylight*) e retornar detalhes (*hosts*, *switches*, portas, etc.) quanto à topologia SDN existente. Posto que esta função não exige a passagem de argumentos, a requisição HTTP com o método GET deve ser feita da seguinte forma:

```
GET /nobi/controller/topology HTTP/1.1
Accept: application/json
```

  Por padrão, a função irá retornar um código no formato JSON [Bray 2014]. Por exemplo, considerando-se a topologia SDN descrita no Capítulo 5 e o controlador *Floodlight* versão 1.2, a requisição anterior retornará a seguinte resposta HTTP:

```
HTTP/1.1 200 OK
Content-type: application/json; charset=utf-8
```

```
1  [
2      {
3          "entityClass": "DefaultEntityClass",
4          "mac": [
5              "00:26:9e:ba:a6:c1"
6          ],
7          "ipv4": [
8              "192.168.24.10"
9          ],
10         "attachmentPoint": [
```

[2] <https://docs.oracle.com/javase/8/docs/api/java/net/HttpURLConnection.html>

```
                {
                    "switchDPID": "00:01:cc:3e:5f:81:ea:01",
                    "port": 33,
                    "errorStatus": null
                }
            ],
            "lastSeen": 1502716476878
        },
        {
            "entityClass": "DefaultEntityClass",
            "mac": [
                "0c:4d:e9:c9:c8:65"
            ],
            "ipv4": [
                "192.168.24.20"
            ],
            "attachmentPoint": [
                {
                    "switchDPID": "00:01:cc:3e:5f:81:ea:01",
                    "port": 49,
                    "errorStatus": null
                }
            ],
            "lastSeen": 1502716489154
        },
        {
            "entityClass": "DefaultEntityClass",
            "mac": [
                "30:f9:ed:c6:72:ca"
            ],
            "ipv4": [
                "192.168.24.24"
            ],
            "attachmentPoint": [
                {
                    "switchDPID": "00:01:cc:3e:5f:81:ea:01",
                    "port": 65,
                    "errorStatus": null
                }
            ],
            "lastSeen": 1502716496182
        }
    ]
```


Código Fonte A.1: Exemplo de código JSON retornado pela URI **`/nobi/controller/topology`** em uma rede SDN controlada pelo *Floodlight* versão 1.2

Por outro lado, ao utilizar o controlador *OpenDaylight Beryllium-SR4* e a mesma topologia, a requisição retornaria uma resposta HTTP com o seguinte código JSON:

```
HTTP/1.1 200 OK
Content-type: application/json; charset=utf-8
```

```
{
  "network-topology":
  {
    "topology":
    [
      {
        "topology-id": "flow:1",
        "node":
        [
          {
            "node-id": "host:0c:4d:e9:c9:c8:65",
            "termination-point":
            [
              {
                "tp-id": "host:0c:4d:e9:c9:c8:65"
              }
            ],
            "host-tracker-service:attachment-points":
            [
              {
                "tp-id": "openflow:391774146382522:20",
                "corresponding-tp": "host:0c:4d:e9:c9:c8:65",
                "active": true
              }
            ],
            "host-tracker-service:id": "0c:4d:e9:c9:c8:65",
            "host-tracker-service:addresses":
            [
              {
                "id": 3,
                "mac": "0c:4d:e9:c9:c8:65",
                "ip": "192.168.24.20",
                "first-seen": 1493386661875,
                "last-seen": 1493388002984
              }
            ]
          },
          {
            "node-id": "openflow:391774146382522",
            "opendaylight-topology-inventory:inventory-node-ref":
              ↪  "/opendaylight-inventory:nodes/opendaylight-inventory:
              ↪  node[opendaylight-inventory:id='openflow:391774146382522']",
            "termination-point":
            [
              {
                "tp-id": "openflow:391774146382522:20",
                "opendaylight-topology-inventory:inventory-node-connector-ref":
                  ↪  "/opendaylight-inventory:nodes/opendaylight-inventory:
                  ↪  node[opendaylight-inventory:id='openflow:391774146382522']
                  ↪  /opendaylight-inventory: node-connector[opendaylight-inventory:
                  ↪  id='openflow:391774146382522:20']"
              },
              {
                "tp-id": "openflow:391774146382522:910",
                "opendaylight-topology-inventory:inventory-node-connector-ref":
                  ↪  "/opendaylight-inventory:nodes/opendaylight-inventory:
                  ↪  node[opendaylight-inventory:id='openflow:391774146382522']
                  ↪  /opendaylight-inventory: node-connector[opendaylight-inventory:
                  ↪  id='openflow:391774146382522:910']"
```

```
50                  },
51                  {
52                    "tp-id": "openflow:391774146382522:10",
53                    "opendaylight-topology-inventory:inventory-node-connector-ref":
                        ↪  "/opendaylight-inventory:nodes/opendaylight-inventory:
                        ↪  node[opendaylight-inventory:id='openflow:391774146382522']
                        ↪  /opendaylight-inventory: node-connector[opendaylight-inventory:
                        ↪  id='openflow:391774146382522:10']"
54                  },
55                  {
56                    "tp-id": "openflow:391774146382522:LOCAL",
57                    "opendaylight-topology-inventory:inventory-node-connector-ref":
                        ↪  "/opendaylight-inventory:nodes/opendaylight-inventory:
                        ↪  node[opendaylight-inventory:id='openflow:391774146382522']
                        ↪  /opendaylight-inventory: node-connector[opendaylight-inventory:
                        ↪  id='openflow:391774146382522:LOCAL']"
58                  },
59                  {
60                    "tp-id": "openflow:391774146382522:24",
61                    "opendaylight-topology-inventory:inventory-node-connector-ref":
                        ↪  "/opendaylight-inventory:nodes/opendaylight-inventory:
                        ↪  node[opendaylight-inventory:id='openflow:391774146382522']
                        ↪  /opendaylight-inventory: node-connector[opendaylight-inventory:
                        ↪  id='openflow:391774146382522:24']"
62                  }
63                ]
64              },
65              {
66                "node-id": "host:00:26:9e:ba:a6:c1",
67                "termination-point":
68                [
69                  {
70                    "tp-id": "host:00:26:9e:ba:a6:c1"
71                  }
72                ],
73                "host-tracker-service:attachment-points":
74                [
75                  {
76                    "tp-id": "openflow:391774146382522:10",
77                    "corresponding-tp": "host:00:26:9e:ba:a6:c1",
78                    "active": true
79                  }
80                ],
81                "host-tracker-service:id": "00:26:9e:ba:a6:c1",
82                "host-tracker-service:addresses":
83                [
84                  {
85                    "id": 2,
86                    "mac": "00:26:9e:ba:a6:c1",
87                    "ip": "192.168.24.10",
88                    "first-seen": 1493384568355,
89                    "last-seen": 1493387524321
90                  }
91                ]
92              },
93              {
94                "node-id": "host:30:f9:ed:c6:72:ca",
95                "termination-point":
96                [
```

```
            {
              "tp-id": "host:30:f9:ed:c6:72:ca"
            }
          ],
          "host-tracker-service:attachment-points":
          [
            {
              "tp-id": "openflow:391774146382522:24",
              "corresponding-tp": "host:30:f9:ed:c6:72:ca",
              "active": true
            }
          ],
          "host-tracker-service:id": "30:f9:ed:c6:72:ca",
          "host-tracker-service:addresses":
          [
            {
              "id": 4,
              "mac": "30:f9:ed:c6:72:ca",
              "ip": "192.168.24.24",
              "first-seen": 1493384578673,
              "last-seen": 1493388571467
            }
          ]
        }
      ],
      "link":
      [
        {
          "link-id": "host:00:26:9e:ba:a6:c1/openflow:391774146382522:10",
          "destination":
          {
            "dest-node": "openflow:391774146382522",
            "dest-tp": "openflow:391774146382522:10"
          },
          "source":
          {
            "source-tp": "host:00:26:9e:ba:a6:c1",
            "source-node": "host:00:26:9e:ba:a6:c1"
          }
        },
        {
          "link-id": "host:0c:4d:e9:c9:c8:65/openflow:391774146382522:20",
          "destination":
          {
            "dest-node": "openflow:391774146382522",
            "dest-tp": "openflow:391774146382522:20"
          },
          "source":
          {
            "source-tp": "host:0c:4d:e9:c9:c8:65",
            "source-node": "host:0c:4d:e9:c9:c8:65"
          }
        },
        {
          "link-id": "openflow:391774146382522:10/host:00:26:9e:ba:a6:c1",
          "destination":
          {
            "dest-node": "host:00:26:9e:ba:a6:c1",
            "dest-tp": "host:00:26:9e:ba:a6:c1"
```

```
156                 },
157                 "source":
158                 {
159                   "source-tp": "openflow:391774146382522:10",
160                   "source-node": "openflow:391774146382522"
161                 }
162               },
163               {
164                 "link-id": "openflow:391774146382522:24/host:30:f9:ed:c6:72:ca",
165                 "destination":
166                 {
167                   "dest-node": "host:30:f9:ed:c6:72:ca",
168                   "dest-tp": "host:30:f9:ed:c6:72:ca"
169                 },
170                 "source":
171                 {
172                   "source-tp": "openflow:391774146382522:24",
173                   "source-node": "openflow:391774146382522"
174                 }
175               },
176               {
177                 "link-id": "host:30:f9:ed:c6:72:ca/openflow:391774146382522:24",
178                 "destination":
179                 {
180                   "dest-node": "openflow:391774146382522",
181                   "dest-tp": "openflow:391774146382522:24"
182                 },
183                 "source":
184                 {
185                   "source-tp": "host:30:f9:ed:c6:72:ca",
186                   "source-node": "host:30:f9:ed:c6:72:ca"
187                 }
188               },
189               {
190                 "link-id": "openflow:391774146382522:20/host:0c:4d:e9:c9:c8:65",
191                 "destination":
192                 {
193                   "dest-node": "host:0c:4d:e9:c9:c8:65",
194                   "dest-tp": "host:0c:4d:e9:c9:c8:65"
195                 },
196                 "source":
197                 {
198                   "source-tp": "openflow:391774146382522:20",
199                   "source-node": "openflow:391774146382522"
200                 }
201               }
202             ]
203           },
204           {
205             "topology-id": "ovsdb:1"
206           }
207         ]
208       }
209   }
```


Código Fonte A.2: Exemplo de código JSON retornado pela URI **`/nobi/controller/topology`** em uma rede SDN controlada pelo *OpenDaylight Beryllium-SR4*

- **`/nobi/controller/active-flows`:** esta função permite ao gerente de redes interagir com o controlador SDN (*Floodlight* ou *OpenDaylight*) para efetuar a listagem detalhada de todas as regras de controle instaladas e ativas na rede. Posto que esta função não exige a passagem de argumentos, uma requisição HTTP simples com o método GET deve ser feita da seguinte forma:

```
GET /nobi/controller/active-flows HTTP/1.1
Accept: application/json
```

Por padrão, a função irá retornar um código no formato JSON [Bray 2014]. A requisição anterior retornará uma resposta HTTP com o código 200 e um arquivo JSON contendo todas as regras de controle ativas na rede SDN. Por exemplo, considerando uma rede SDN e o controlador *Floodlight* versão 1.2 com regras para para tratar pacotes ARP, LLDP e encaminhar tráfego UDP marcado para uma fila de serviço prioritário, uma resposta HTTP poderia ser retornada com o seguinte código JSON abaixo:

```
HTTP/1.1 200 OK
Content-type: application/json; charset=utf-8
```

```
{
  "00:01:cc:3e:5f:81:ea:01":
  [
    {
      "UDP-HOST10-TO-HOST24-PORT5001-SETQUEUE4":
      {
        "version": "OF_13",
        "command": "ADD",
        "cookie": "45035996376605075",
        "priority": "30",
        "idleTimeoutSec": "0",
        "hardTimeoutSec": "0",
        "outPort": "any",
        "flags": "1",
        "cookieMask": "0",
        "outGroup": "any",
        "match":
        {
          "eth_dst": "30:f9:ed:c6:72:ca",
          "eth_src": "0c:26:9e:ba:a6:c1",
          "eth_type": "2048",
          "ip_proto": "17",
          "ipv4_src": "192.168.24.10",
          "ipv4_dst": "192.168.24.24",
          "udp_dst": "5001",
          "active": "true",
```

```
            "ip_dscp": "34"
          },
          "instructions":
          {
            "instruction_apply_actions":
            {
              "actions": "set_queue=4"
            }
          }
        }
      },
      {
        "ARP-TO-CONTROLLER":
        {
          "version": "OF_13",
          "command": "ADD",
          "cookie": "45035996779185181",
          "priority": "5",
          "idleTimeoutSec": "0",
          "hardTimeoutSec": "0",
          "outPort": "any",
          "flags": "1",
          "cookieMask": "0",
          "outGroup": "any",
          "match":
          {
            "eth_type": "2054"
          },
          "instructions":
          {
            "instruction_apply_actions":
            {
              "actions": "output=controller"
            }
          }
        }
      },
      {
        "LLDP-TO-CONTROLLER":
        {
          "version": "OF_13",
          "command": "ADD",
          "cookie": "45035997113348908",
          "priority": "5",
          "idleTimeoutSec": "0",
          "hardTimeoutSec": "0",
          "outPort": "any",
          "flags": "1",
          "cookieMask": "0",
          "outGroup": "any",
          "match":
          {
            "eth_type": "35020"
          },
          "instructions":
          {
            "instruction_apply_actions":
            {
              "actions": "output=controller"
```

```
86              }
87            }
88          }
89        },
90        {
91          "SEND-ALL-TO-CONTROLLER":
92          {
93            "version": "OF_13",
94            "command": "ADD",
95            "cookie": "45035999440417732",
96            "priority": "0",
97            "idleTimeoutSec": "0",
98            "hardTimeoutSec": "0",
99            "outPort": "any",
100           "flags": "1",
101           "cookieMask": "0",
102           "outGroup": "any",
103           "match": {},
104           "instructions":
105           {
106             "instruction_apply_actions":
107             {
108               "actions": "output=controller"
109             }
110           }
111         }
112       }
113     ]
114   }
```

Código Fonte A.3: Exemplo de código JSON retornado pela URI **`/nobi/controller/active-flows`** em uma rede SDN controlada pelo *Floodlight* versão 1.2

Usando o controlador *OpenDaylight Beryllium-SR4* nas mesmas condições do exemplo anterior, a função também retornará uma resposta HTTP com a listagem das regras de controle ativas na SDN, conforme é mostrado no arquivo JSON abaixo:

```
HTTP/1.1 200 OK
Content-type: application/json; charset=utf-8
```

```
1   {
2     "flow-node-inventory:table":
3     [
4       {
5         "id": 0,
6         "flow":
7         [
8           {
9             "id": "99",
10            "match":
11            {
12              "ipv4-destination": "192.168.24.24/24",
```

```
            "ipv4-source": "192.168.24.10/24",
            "ethernet-match":
            {
              "ethernet-type":
              {
                "type": 2048
              }
            },
            "tcp-destination-port": 5001,
            "ip-match":
            {
              "ip-protocol": 17,
              "ip-dscp": 34
            }
          },
          "hard-timeout": 0,
          "idle-timeout": 0,
          "flags": "SEND_FLOW_REM",
          "priority": 30,
          "table_id": 0,
          "opendaylight-flow-statistics:flow-statistics":
          {
            "packet-count": 1167233,
            "byte-count": 18446744073709551615,
            "duration":
            {
              "nanosecond": 4294967295,
              "second": 1905
            }
          },
          "cookie": 0,
          "instructions":
          {
            "instruction":
            [
              {
                "order": 0,
                "apply-actions":
                {
                  "action":
                  [
                    {
                      "order": 0,
                      "set-queue-action":
                      {
                        "queue-id": 4
                      }
                    }
                  ]
                }
              }
            ]
          }
        },
        {
          "id": "#UF$TABLE*0-13",
          "match":
          {
            "ethernet-match":
```

```
            {
              "ethernet-type":
              {
                "type": 35020
              }
            }
          },
          "hard-timeout": 0,
          "idle-timeout": 0,
          "flags": "",
          "priority": 5,
          "table_id": 0,
          "opendaylight-flow-statistics:flow-statistics":
          {
            "packet-count": 0,
            "byte-count": 18446744073709551615,
            "duration":
            {
              "nanosecond": 4294967295,
              "second": 4362
            }
          },
          "cookie": 3098476543630901253,
          "instructions":
          {
            "instruction":
            [
              {
                "order": 0,
                "apply-actions":
                {
                  "action":
                  [
                    {
                      "order": 0,
                      "output-action":
                      {
                        "output-node-connector": "CONTROLLER",
                        "max-length": 65535
                      }
                    }
                  ]
                }
              }
            ]
          }
        },
        {
          "id": "#UF$TABLE*0-35",
          "match":
          {
            "ethernet-match":
            {
              "ethernet-source":
              {
                "address": "0C:26:9E:BA:A6:C1"
              },
              "ethernet-destination":
              {
```

```
                "address": "30:F9:ED:C6:72:CA"
              }
            }
          },
          "hard-timeout": 600,
          "idle-timeout": 300,
          "flags": "",
          "priority": 10,
          "table_id": 0,
          "opendaylight-flow-statistics:flow-statistics":
          {
            "packet-count": 71,
            "byte-count": 18446744073709551615,
            "duration":
            {
              "nanosecond": 4294967295,
              "second": 335
            }
          },
          "cookie": 3026418949592973339,
          "instructions":
          {
            "instruction":
            [
              {
                "order": 0,
                "apply-actions":
                {
                  "action":
                  [
                    {
                      "order": 0,
                      "output-action":
                      {
                        "output-node-connector": "24",
                        "max-length": 65535
                      }
                    }
                  ]
                }
              }
            ]
          }
        },
        {
          "id": "#UF$TABLE*0-12",
          "match":
          {
            "ethernet-match":
            {
              "ethernet-type":
              {
                "type": 2054
              }
            }
          },
          "hard-timeout": 0,
          "idle-timeout": 0,
          "flags": "",
```

```
            "priority": 5,
            "table_id": 0,
            "opendaylight-flow-statistics:flow-statistics":
            {
              "packet-count": 86,
              "byte-count": 18446744073709551615,
              "duration":
              {
                "nanosecond": 4294967295,
                "second": 4362
              }
            },
            "cookie": 3098476543630901253,
            "instructions":
            {
              "instruction":
              [
                {
                  "order": 0,
                  "apply-actions":
                  {
                    "action":
                    [
                      {
                        "order": 0,
                        "output-action":
                        {
                          "output-node-connector": "CONTROLLER",
                          "max-length": 65535
                        }
                      }
                    ]
                  }
                }
              ]
            }
          },
          {
            "id": "#UF$TABLE*0-34",
            "match":
            {
              "ethernet-match":
              {
                "ethernet-source":
                {
                  "address": "30:F9:ED:C6:72:CA"
                },
                "ethernet-destination":
                {
                  "address": "0C:26:9E:BA:A6:C1"
                }
              }
            },
            "hard-timeout": 600,
            "idle-timeout": 300,
            "flags": "",
            "priority": 10,
            "table_id": 0,
            "opendaylight-flow-statistics:flow-statistics":
```

```
            {
              "packet-count": 10,
              "byte-count": 18446744073709551615,
              "duration":
              {
                "nanosecond": 4294967295,
                "second": 335
              }
            },
            "cookie": 3026418949592973338,
            "instructions":
            {
              "instruction":
              [
                {
                  "order": 0,
                  "apply-actions":
                  {
                    "action":
                    [
                      {
                        "order": 0,
                        "output-action":
                        {
                          "output-node-connector": "10",
                          "max-length": 65535
                        }
                      }
                    ]
                  }
                }
              ]
            }
          }
        ],
        "flow-hash-id-map":
        [
          {
            "hash": "Match [_ethernetMatch=EthernetMatch [_ethernetType=EthernetType
              ↪  [_type=EtherType [_value=35020], augmentation=[]], augmentation=[]],
              ↪  augmentation=[]]10030984765436309012 53",
            "flow-id": "#UF$TABLE*0-13"
          },
          {
            "hash": "Match [_ethernetMatch=EthernetMatch
              ↪  [_ethernetDestination=EthernetDestination [_address=MacAddress
              ↪  [_value=30:F9:ED:C6:72:CA], augmentation=[]],
              ↪  _ethernetSource=EthernetSource [_address=MacAddress
              ↪  [_value=0C:26:9E:BA:A6:C1], augmentation=[]], augmentation=[]],
              ↪  augmentation=[]]10302641894959297333 9",
            "flow-id": "#UF$TABLE*0-35"
          },
          {
```

```
295             "hash": "Match [_ethernetMatch=EthernetMatch [_ethernetType=EthernetType
                  ↪  [_type=EtherType [_value=2048], augmentation=[]], augmentation=[]],
                  ↪  _ipMatch=IpMatch [_ipDscp=Dscp [_value=34], _ipProtocol=17,
                  ↪  augmentation=[]], _layer3Match=Ipv4Match [_ipv4Destination=Ipv4Prefix
                  ↪  [_value=192.168.24.24/24], _ipv4Source=Ipv4Prefix
                  ↪  [_value=192.168.24.10/24], augmentation=[]], _layer4Match=TcpMatch
                  ↪  [_tcpDestinationPort=PortNumber [_value=5001], augmentation=[]],
                  ↪  augmentation=[]]300",
296             "flow-id": "99"
297           },
298           {
299             "hash": "Match [_ethernetMatch=EthernetMatch
                  ↪  [_ethernetDestination=EthernetDestination [_address=MacAddress
                  ↪  [_value=0C:26:9E:BA:A6:C1], augmentation=[]],
                  ↪  _ethernetSource=EthernetSource [_address=MacAddress
                  ↪  [_value=30:F9:ED:C6:72:CA], augmentation=[]], augmentation=[]],
                  ↪  augmentation=[]]10302641894959297338",
300             "flow-id": "#UF$TABLE*0-34"
301           },
302           {
303             "hash": "Match [_ethernetMatch=EthernetMatch [_ethernetType=EthernetType
                  ↪  [_type=EtherType [_value=2054], augmentation=[]], augmentation=[]],
                  ↪  augmentation=[]]13098476543630901253",
304             "flow-id": "#UF$TABLE*0-12"
305           }
306         ],
307         "opendaylight-flow-table-statistics:flow-table-statistics":
308         {
309           "packets-matched": 1167400,
310           "packets-looked-up": 1167859,
311           "active-flows": 5
312         }
313       }
314     ]
315   }
```

Código Fonte A.4: Exemplo de código JSON retornado pela URI **`/nobi/controller/active-flows`** em uma rede SDN controlada pelo *OpenDaylight Beryllium-SR4*

- **`/nobi/controller/static-flows`:** esta função é capaz de listar apenas as regras de controle instaladas pelo gerente na rede SDN, em vez daquelas instaladas (de forma automática) pelo controlador. Não é exigida do cliente da função a passagem de parâmetros e a requisição HTTP deve ser levada a efeito da seguinte forma:

```
GET /nobi/controller/static-flows HTTP/1.1
Accept: application/json
```

Por padrão, a função irá retornar um código no formato JSON [Bray 2014]. A requisição anterior retornará uma resposta HTTP com o código 200 e um arquivo JSON com todas

as regras não instaladas pelo controlador na rede. No exemplo abaixo, considere que a resposta HTTP mostra duas regras aplicadas pelo gerente no *Floodlight* versão 1.2 tendo em vista a comunicação do tráfego entre portas *OpenFlow*, como segue:

```
HTTP/1.1 200 OK
Content-type: application/json; charset=utf-8
```


```
{
  "00:01:cc:3e:5f:81:ea:01":
  [
    {
      "FROM-OFPORT-65-TO-33":
      {
        "version": "OF_13",
        "command": "ADD",
        "cookie": "45035996574880701",
        "priority": "10",
        "idleTimeoutSec": "100",
        "hardTimeoutSec": "300",
        "flags": "1",
        "cookieMask": "0",
        "outGroup": "any",
        "match":
        {
          "in_port": "65"
        },
        "instructions":
        {
          "instruction_apply_actions":
          {
            "actions": "output=33"
          }
        }
      }
    },
    {
      "FROM-OFPORT-33-TO-65":
      {
        "version": "OF_13",
        "command": "ADD",
        "cookie": "45035996931017749",
        "priority": "10",
        "idleTimeoutSec": "100",
        "hardTimeoutSec": "300",
        "flags": "1",
        "cookieMask": "0",
        "outGroup": "any",
        "match":
        {
          "in_port": "33"
        },
        "instructions":
        {
          "instruction_apply_actions":
          {
```

```
49                "actions": "output=65"
50              }
51            }
52          }
53        }
54      ]
55    }
```

Código Fonte A.5: Exemplo de código JSON retornado pela URI **`/nobi/controller/static-flows`** em uma rede SDN controlada pelo *Floodlight* versão 1.2

Utilizando o controlador *OpenDaylight Beryllium-SR4* nas mesmas condições do exemplo anterior, a função ainda retornará uma resposta HTTP com a listagem das regras de controle instaladas pelo gerente na rede, conforme é mostrado no arquivo JSON abaixo:

```
HTTP/1.1 200 OK
Content-type: application/json; charset=utf-8
```

```
1   {
2     "flow-node-inventory:table":
3     [
4       {
5         "id": 0,
6         "flow":
7         [
8           {
9             "id": "6533",
10            "hard-timeout": 300,
11            "match":
12            {
13              "in-port": "65"
14            },
15            "idle-timeout": 100,
16            "priority": 10,
17            "table_id": 0,
18            "flow-name": "FROM-OFPORT-65-TO-33",
19            "instructions":
20            {
21              "instruction":
22              [
23                {
24                  "order": 0,
25                  "apply-actions":
26                  {
27                    "action":
28                    [
29                      {
30                        "order": 0,
31                        "output-action":
32                        {
33                          "output-node-connector": "33"
34                        }
```

```
                        }
                      ]
                    }
                  }
                ]
              }
            },
            {
              "id": "3365",
              "hard-timeout": 300,
              "match":
              {
                "in-port": "33"
              },
              "idle-timeout": 100,
              "priority": 10,
              "table_id": 0,
              "flow-name": "FROM-OFPORT-33-TO-65",
              "instructions":
              {
                "instruction":
                [
                  {
                    "order": 0,
                    "apply-actions":
                    {
                      "action":
                      [
                        {
                          "order": 0,
                          "output-action":
                          {
                            "output-node-connector": "65"
                          }
                        }
                      ]
                    }
                  }
                ]
              }
            }
          ]
        }
    ]
}
```


Código Fonte A.6: Exemplo de código JSON retornado pela URI **`/nobi/controller/static-flows`** em uma rede SDN controlada pelo *OpenDaylight Beryllium-SR4*

- **`/nobi/controller/switches`:** esta função tem por objetivo comunicar-se com o controlador e retornar informações acerca dos *switches* existentes no plano de dados SDN. Não é exigida do cliente da função a passagem de parâmetros e uma requisição HTTP simples pode ser levada a efeito da seguinte forma:

```
GET /nobi/controller/switches HTTP/1.1
Accept: application/json
```

Por padrão, a função irá retornar um código no formato JSON [Bray 2014]. Por exemplo, considerando-se a topologia SDN descrita no Capítulo 5 e o *Floodlight* versão 1.2, a requisição anterior retornará a seguinte resposta HTTP com código 200:

```
HTTP/1.1 200 OK
Content-type: application/json; charset=utf-8
```

```
[
    {
        "inetAddress": "/192.168.1.254:47237",
        "connectedSince": 1502990331563,
        "switchDPID": "00:01:cc:3e:5f:81:ea:01"
    }
]
```

Código Fonte A.7: Exemplo de código JSON retornado pela URI **`/nobi/controller/switches`** em uma rede SDN controlada pelo *Floodlight* versão 1.2

Por outro lado, ao utilizar o controlador *OpenDaylight Beryllium-SR4* e a mesma topologia, a requisição retornará uma resposta HTTP com o seguinte código JSON:

```
HTTP/1.1 200 OK
Content-type: application/json; charset=utf-8
```

```
{
  "nodes":
  {
    "node":
    [
      {
        "id": "openflow:391774146382522",
        "flow-node-inventory:switch-features":
        {
          "capabilities":
          [
            "flow-node-inventory:flow-feature-capability-port-blocked",
            "flow-node-inventory:flow-feature-capability-queue-stats",
            "flow-node-inventory:flow-feature-capability-table-stats",
            "flow-node-inventory:flow-feature-capability-flow-stats",
            "flow-node-inventory:flow-feature-capability-port-stats"
          ],
          "max_buffers": 1024,
          "max_tables": 1
```

```
        },
        "flow-node-inventory:ip-address": "192.168.1.254",
        "flow-node-inventory:manufacturer": "HP",
        "flow-node-inventory:software": "Comware Software, Version 7.1.045",
        "flow-node-inventory:serial-number": "BR58HCG086",
        "flow-node-inventory:description": "",
        "flow-node-inventory:hardware": "HP 5130-24G-4SFP+ EI Brazil Switch",
        "node-connector":
        [
          {
            "id": "openflow:391774146382522:LOCAL",
            "flow-node-inventory:configuration": "",
            "flow-node-inventory:name": "OFPP_LOCAL",
            "flow-node-inventory:current-speed": 0,
            "flow-node-inventory:current-feature": "",
            "flow-node-inventory:hardware-address": "64:51:06:D3:CA:BA",
            "flow-node-inventory:peer-features": "",
            "flow-node-inventory:maximum-speed": 0,
            "flow-node-inventory:advertised-features": "",
            "flow-node-inventory:state":
            {
              "link-down": false,
              "blocked": false,
              "live": true
            },
            "flow-node-inventory:supported": "",
            "flow-node-inventory:port-number": "LOCAL"
          },
          {
            "id": "openflow:391774146382522:24",
            "flow-node-inventory:configuration": "",
            "flow-node-inventory:name": "GE1/0/24",
            "flow-node-inventory:current-speed": 1000000,
            "flow-node-inventory:current-feature": "one-gb-fd",
            "flow-node-inventory:hardware-address": "64:51:06:D3:CA:FA",
            "flow-node-inventory:peer-features": "",
            "flow-node-inventory:maximum-speed": 1000000,
            "flow-node-inventory:advertised-features": "one-gb-fd",
            "flow-node-inventory:state":
            {
              "link-down": false,
              "blocked": false,
              "live": true
            },
            "flow-node-inventory:supported": "one-gb-hd one-gb-fd",
            "flow-node-inventory:port-number": "24",
            "address-tracker:addresses":
            [
              {
                "id": 4,
                "mac": "30:f9:ed:c6:72:ca",
                "ip": "192.168.24.24",
                "first-seen": 1493384578673,
                "last-seen": 1493388571467
              }
            ]
          },
          {
            "id": "openflow:391774146382522:20",
```

```
            "flow-node-inventory:configuration": "",
            "flow-node-inventory:name": "GE1/0/20",
            "flow-node-inventory:current-speed": 1000000,
            "flow-node-inventory:current-feature": "one-gb-fd",
            "flow-node-inventory:hardware-address": "64:51:06:D3:CA:F6",
            "flow-node-inventory:peer-features": "",
            "flow-node-inventory:maximum-speed": 1000000,
            "flow-node-inventory:advertised-features": "one-gb-fd",
            "flow-node-inventory:state":
            {
              "link-down": true,
              "blocked": true,
              "live": false
            },
            "flow-node-inventory:supported": "one-gb-hd one-gb-fd",
            "flow-node-inventory:port-number": "20",
            "address-tracker:addresses":
            [
              {
                "id": 3,
                "mac": "0c:4d:e9:c9:c8:65",
                "ip": "192.168.24.20",
                "first-seen": 1493385115552,
                "last-seen": 1493386661878
              }
            ]
          },
          {
            "id": "openflow:391774146382522:10",
            "flow-node-inventory:configuration": "",
            "flow-node-inventory:name": "GE1/0/10",
            "flow-node-inventory:current-speed": 100000,
            "flow-node-inventory:current-feature": "hundred-mb-fd",
            "flow-node-inventory:hardware-address": "64:51:06:D3:CA:EC",
            "flow-node-inventory:peer-features": "",
            "flow-node-inventory:maximum-speed": 1000000,
            "flow-node-inventory:advertised-features": "hundred-mb-fd",
            "flow-node-inventory:state":
            {
              "link-down": false,
              "blocked": false,
              "live": true
            },
            "flow-node-inventory:supported": "hundred-mb-fd hundred-mb-hd",
            "flow-node-inventory:port-number": "10",
            "address-tracker:addresses":
            [
              {
                "id": 2,
                "mac": "00:26:9e:ba:a6:c1",
                "ip": "192.168.24.10",
                "first-seen": 1493384568355,
                "last-seen": 1493387524321
              }
            ]
          },
          {
            "id": "openflow:391774146382522:910",
            "flow-node-inventory:configuration": "",
```

```
138                 "flow-node-inventory:name": "Vlan24",
139                 "flow-node-inventory:current-speed": 0,
140                 "flow-node-inventory:current-feature": "",
141                 "flow-node-inventory:hardware-address": "64:51:06:D3:CA:DB",
142                 "flow-node-inventory:peer-features": "",
143                 "flow-node-inventory:maximum-speed": 0,
144                 "flow-node-inventory:advertised-features": "",
145                 "flow-node-inventory:state":
146                 {
147                   "link-down": false,
148                   "blocked": false,
149                   "live": true
150                 },
151                 "flow-node-inventory:supported": "",
152                 "flow-node-inventory:port-number": "910"
153               }
154             ]
155           }
156       ]
157     }
158 }
```

Código Fonte A.8: Exemplo de código JSON retornado pela URI **`/nobi/controller/switches`** em uma rede SDN controlada pelo *OpenDaylight Beryllium-SR4*

- **`/nobi/controller/push-flow`:** esta é uma função genérica que permite ao gerente instalar instruções na rede, garantindo-lhe a liberdade necessária para elaborar regras de controle e acomodá-las na SDN—sobretudo quando estas não forem suportadas por funções nativas da interface NoBI. Neste caso, deve-se escrever a regra utilizando as estruturas (*tags*) e o formato de arquivo padrão suportados pelo controlador-alvo. O gerente deve fornecer como parâmetro o corpo da regra de controle e uma indicação do formato do arquivo no cabeçalho HTTP, quando efetuar a requisição do tipo POST. Se, por um lado, as regras de controle são repassadas ao *Floodlight* no formato JSON, por outro, no controlador *OpenDaylight*, estas devem ser descritas no formato XML. Por exemplo, para instalar no *Floodlight* versão 1.2 uma regra capaz de tratar fluxos de pacotes TCP de chegada na porta 33 do equipamento *OpenFlow*, originados do IP 10.0.0.1, destinados à porta 8080 do *host* 10.0.0.2, e que serão encaminhados pela porta 65, pode-se fazer a seguinte requisição HTTP:

```
GET /nobi/controller/static-flows HTTP/1.1
Accept: application/json
```

```
1 {
2   "switch":"00:01:cc:3e:5f:81:ea:01",
```

```
    "name":"TCP-HOST1-TO-HOST2-DESTPORT-8080",
    "priority":"10",
    "idle_timeout": "40",
    "hard_timeout": "60",
    "active":"true",
    "in_port": "33",
    "eth_src": "00:00:00:00:00:01",
    "eth_dst": "00:00:00:00:00:02",
    "eth_type": "2048",
    "ip_proto": "6",
    "tp_dst": "8080",
    "ipv4_src": "10.0.0.1",
    "ipv4_dst": "10.0.0.2",
    "actions":"output=65"
}
```

No caso do *OpenDaylight*, deve-se descrever a regra anterior usando *tags* suportadas por este controlador no formato XML, como segue:

```
    HTTP/1.1 200 OK
    Content-type: application/json; charset=utf-8
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <output-action>
                        <output-node-connector>65</output-node-connector>
                    </output-action>
                    <order>0</order>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
    <table_id>0</table_id>
    <id>17</id>
    <match>
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
            <ethernet-source>
                <address>00:00:00:00:00:01</address>
            </ethernet-source>
            <ethernet-destination>
                <address>00:00:00:00:00:02</address>
            </ethernet-destination>
        </ethernet-match>
        <ipv4-source>10.0.0.1/24</ipv4-source>
        <ipv4-destination>10.0.0.2/24</ipv4-destination>
        <ip-match>
            <ip-protocol>6</ip-protocol>
```

```
34          </ip-match>
35          <tcp-destination-port>8080</tcp-destination-port>
36          <in-port>33</in-port>
37      </match>
38      <hard-timeout>60</hard-timeout>
39      <idle-timeout>40</idle-timeout>
40      <flow-name>"TCP-HOST1-TO-HOST2-DESTPORT-8080"</flow-name>
41      <priority>10</priority>
42  </flow>
```

Quando uma entidade é criada no servidor com o método POST, a especificação do protocolo HTTP [Fielding *et al.* 1999] informa que o código HTTP 201 ("*Created*") deve ser retornado com, opcionalmente, uma representação da entidade na resposta. Optou-se, pois, por retornar uma resposta HTTP contendo todo o código da regra recém-instalada no controlador (*Floodlight* ou *OpenDaylight*), juntamente com o código HTTP 201, conforme é mostrado nos exemplos abaixo:

```
HTTP/1.1 201 Created
Content-type: application/json; charset=utf-8
```

```
1  {
2    "switch":"00:01:cc:3e:5f:81:ea:01",
3    "name":"TCP-HOST1-TO-HOST2-DESTPORT-8080",
4    "priority":"10",
5    "idle_timeout": "40",
6    "hard_timeout": "60",
7    "active":"true",
8    "in_port": "33",
9    "eth_src": "00:00:00:00:00:01",
10   "eth_dst": "00:00:00:00:00:02",
11   "eth_type": "2048",
12   "ip_proto": "6",
13   "tp_dst": "8080",
14   "ipv4_src": "10.0.0.1",
15   "ipv4_dst": "10.0.0.2",
16   "actions":"output=65"
17 }
```

Código Fonte A.9: Exemplo de código JSON retornado pela URI **`/nobi/controller/push-flow`** em uma rede SDN controlada pelo *Floodlight* versão 1.2

```
HTTP/1.1 201 Created
Content-type: application/xml; charset=utf-8
```

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <flow xmlns="urn:opendaylight:flow:inventory">
3      <instructions>
```

```
4           <instruction>
5               <order>0</order>
6               <apply-actions>
7                   <action>
8                       <output-action>
9                           <output-node-connector>65</output-node-connector>
10                      </output-action>
11                      <order>0</order>
12                  </action>
13              </apply-actions>
14          </instruction>
15      </instructions>
16      <table_id>0</table_id>
17      <id>17</id>
18      <match>
19          <ethernet-match>
20              <ethernet-type>
21                  <type>2048</type>
22              </ethernet-type>
23              <ethernet-source>
24                  <address>00:00:00:00:00:01</address>
25              </ethernet-source>
26              <ethernet-destination>
27                  <address>00:00:00:00:00:02</address>
28              </ethernet-destination>
29          </ethernet-match>
30          <ipv4-source>10.0.0.1/24</ipv4-source>
31          <ipv4-destination>10.0.0.2/24</ipv4-destination>
32          <ip-match>
33              <ip-protocol>6</ip-protocol>
34          </ip-match>
35          <tcp-destination-port>8080</tcp-destination-port>
36          <in-port>33</in-port>
37      </match>
38      <hard-timeout>60</hard-timeout>
39      <idle-timeout>40</idle-timeout>
40      <flow-name>"TCP-HOST1-TO-HOST2-DESTPORT-8080"</flow-name>
41      <priority>10</priority>
42  </flow>
```

Código Fonte A.10: Exemplo de código XML retornado pela URI **`/nobi/controller/push-flow`** em uma rede SDN controlada pelo *OpenDaylight Beryllium-SR4*

- **`/nobi/controller/send-to-controller`:** a especificação do protocolo *OpenFlow* versão 1.3.0 [ONF 2012] afirma que quando um fluxo de pacotes não pode ser tratado por nenhuma das regras de decisão existentes no equipamento de rede, estes serão descartados (*dropped*), por padrão. A mesma especificação cita, porém, que este comportamento pode ser modificado. Neste contexto, a abordagem NoBI oferece ao gerente uma função capaz de instalar uma regra para, por padrão, encaminhar os pacotes que não tiveram uma regra correspondente para o controlador. Esta função não exige a

passagem de parâmetros. Apenas uma simples requisição HTTP do tipo POST deve ser feita da seguinte forma:

```
HTTP/1.1 201 Created
Content-type: application/xml; charset=utf-8
```

Quando uma entidade é criada no servidor com o método POST, a especificação do protocolo HTTP [Fielding *et al.* 1999] informa que o código HTTP 201 ("*Created*") deve ser retornado com, opcionalmente, uma representação da entidade na resposta. Optou-se, pois, por retornar uma resposta HTTP contendo todo o código da regra recém-instalada no controlador (*Floodlight* ou *OpenDaylight*), juntamente com o código HTTP 201, conforme é mostrado nos exemplos abaixo:

```
HTTP/1.1 201 Created
Content-type: application/json; charset=utf-8
```

```
{
    "switch":"00:01:cc:3e:5f:81:ea:01",
    "name":"SEND-ALL-TO-CONTROLLER",
    "cookie":"0",
    "priority": "0",
    "idleTimeoutSec": "0",
    "hardTimeoutSec": "0",
    "active":"true",
    "actions": "output=controller"
}
```

Código Fonte A.11: Exemplo de código JSON retornado pela URI **`/nobi/controller/send-to-controller`** em uma rede SDN controlada pelo *Floodlight* versão 1.2

```
HTTP/1.1 201 Created
Content-type: application/xml; charset=utf-8
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <flow-name>SEND-ALL-TO-CONTROLLER</flow-name>
    <id>0</id>
    <table_id>0</table_id>
    <priority>0</priority>
    <hard-timeout>0</hard-timeout>
    <idle-timeout>0</idle-timeout>
    <instructions>
        <instruction>
            <order>0</order>
```

```
12              <apply-actions>
13                  <action>
14                      <order>0</order>
15                      <output-action>
16                          <output-node-connector>CONTROLLER</output-node-connector>
17                          <max-length>65535</max-length>
18                      </output-action>
19                  </action>
20              </apply-actions>
21          </instruction>
22      </instructions>
23  </flow>
```

Código Fonte A.12: Exemplo de código XML retornado pela URI **`/nobi/controller/send-to-controller`** em uma rede SDN controlada pelo *OpenDaylight Beryllium-SR4*

- **`/nobi/controller/delete-flow`:** esta função permite remover uma regra de controle existente no controlador (*Floodlight* ou *OpenDaylight*). O cliente da função deve fornecer como parâmetro o identificador da regra `<id>` a ser removida no formato JSON, quando efetuar a requisição HTTP do tipo DELETE. Por exemplo, para excluir do controlador a regra identificada por `ABC123`, o cliente deve efetuar a requisição HTTP contendo o seguinte código JSON como parâmetro:

```
DELETE /nobi/controller/delete-flow HTTP/1.1
Accept: application/json
Content-type: application/json
```

```
1  {"id":"ABC123"}
```

  Para esta função, alinhado à especificação do protocolo HTTP [Fielding *et al.* 1999], buscou-se retornar o código HTTP 200 ("OK"), se a resposta incluir a entidade que representa o resultado (i.e., a regra recém-removida do controlador), ou o código 204 ("*No Content*"), em caso contrário.

- **`/nobi/controller/create-qos-path`:** considerando que diferentes classes de tráfego podem ser tratadas de formas diferentes na rede (em vez de um único serviço de melhor esforço), esta função permite aplicar uma regra no controlador para programar a SDN e efetuar a reserva de largura de banda na comunicação entre pontos (origem/-destino) da rede. Particularmente, a partir da classificação de pacotes, o gerente poderá, em alto nível, solicitar o encaminhamento do tráfego marcado para uma fila de serviço

e garantir uma taxa de vazão mínima. Por exemplo, pacotes pertencentes a uma aplicação multimídia poderiam ter prioridade sobre outros pacotes para receber um serviço diferenciado na rede. Neste cenário, esta função se mostra bastante útil no mapeamento do tráfego multimídia para uma fila de serviço e auxilia o gerente na configuração e estabelecimento de QoS em alto nível na SDN. A fila para a qual o tráfego será direcionado é obtida com base na classe de serviço do tráfego (um parâmetro para a função) e a configuração padrão de mapeamento (DSCP $\Longrightarrow$ Fila) do equipamento *OpenFlow* que receberá a solicitação. Os seguintes parâmetros devem ser fornecidos para a função: i) o endereço IP da origem `<ipv4_src>`; ii) o IP de destino `<ipv4_dst>`; iii) um identificador de regra `<id>` único, escolhido pelo gerente; iv) o código DSCP [Nichols *et al.* 1998] indicando a classe de serviço do tráfego que terá reserva de banda controlada `<ip_dscp>`; v) o tipo do protocolo `<eth_type>` [IANA 2017a]; vi) o identificador do protocolo da camada superior `<ip_proto>` [IANA 2017][3]; e vii) a reserva (mínima) de largura de banda desejada `<min_bandwidth_kbps>`, em kbps. O cliente da função deve fornecer todos estes parâmetros e utilizar o formato padrão JSON [Bray 2014], quando efetuar a requisição HTTP do tipo POST. Por exemplo, para instalar no controlador (*Floodlight* ou *OpenDaylight*) uma regra identificada por `<id>=99` que visa a reserva de largura de banda mínima de 12 Mbps apenas para o tráfego UDP marcado com DSCP 34 (i.e., $AF41$) na comunicação entre dois *hosts* da rede com endereços de origem `<ipv4_src>=192.168.24.10` e destino `<ipv4_dst>=192.168.24.24`, deve-se fazer a seguinte requisição HTTP:

```
POST /nobi/controller/create-qos-path HTTP/1.1
Accept: application/json
Content-type: application/json
```

```
{
  "id":"99",
  "eth_type": "2048",
  "ip_proto": "17",
  "ipv4_src": "192.168.24.10",
  "ipv4_dst": "192.168.24.24",
  "ip_dscp": "34",
  "min_bandwidth_kbps": "12000"
}
```

[3] Por exemplo, um valor 6 indica que a porção de dados do datagrama IP será passada ao TCP, enquanto um valor 17 indica que os dados serão passados ao UDP.

Quando uma entidade é criada no servidor com o método POST, a especificação do protocolo HTTP [Fielding *et al.* 1999] informa que o código HTTP 201 ("*Created*") deve ser retornado com, opcionalmente, uma representação da entidade na resposta. Optou-se, pois, por retornar uma resposta HTTP contendo todo o código da regra recém-instalada no controlador (*Floodlight* ou *OpenDaylight*), juntamente com o código HTTP 201, conforme é mostrado nos exemplos abaixo:

```
HTTP/1.1 201 Created
Content-type: application/json; charset=utf-8
```

```
{
  "00:01:cc:3e:5f:81:ea:01":
  [
    {
      "99":
      {
        "version": "OF_13",
        "command": "ADD",
        "cookie": "45035996376605075",
        "priority": "30",
        "idleTimeoutSec": "0",
        "hardTimeoutSec": "0",
        "outPort": "any",
        "flags": "1",
        "cookieMask": "0",
        "outGroup": "any",
        "match":
        {
          "eth_dst": "30:f9:ed:c6:72:ca",
          "eth_src": "0c:26:9e:ba:a6:c1",
          "eth_type": "2048",
          "ip_proto": "17",
          "ipv4_src": "192.168.24.10",
          "ipv4_dst": "192.168.24.24",
          "active": "true",
          "ip_dscp": "34"
        },
        "instructions":
        {
          "instruction_apply_actions":
          {
            "actions": "set_queue=4"
          }
        }
      }
    }
  ]
}
```

Código Fonte A.13: Exemplo de código JSON retornado pela URI **`/nobi/controller/create-qos-path`** em uma rede SDN controlada pelo *Floodlight* versão 1.2

```
HTTP/1.1 201 Created
Content-type: application/json; charset=utf-8
```


```
{
  "flow-node-inventory:table":
  [
    {
      "id": 0,
      "flow":
      [
        {
          "id": "99",
          "match":
          {
            "ethernet-match":
            {
              "ethernet-type":
              {
                "type": 2048
              }
            },
            "ipv4-destination": "192.168.24.24/24",
            "ipv4-source": "192.168.24.10/24",
            "ip-match":
            {
              "ip-protocol": 17,
              "ip-dscp": 34
            }
          },
          "hard-timeout": 0,
          "idle-timeout": 300,
          "flags": "SEND_FLOW_REM",
          "priority": 30,
          "table_id": 0,
          "instructions":
          {
            "instruction":
            [
              {
                "order": 0,
                "apply-actions":
                {
                  "action":
                  [
                    {
                      "order": 0,
                      "set-queue-action":
                      {
                        "queue-id": 4,
                        "queue": "4"
                      }
                    }
                  ]
                }
              }
            ]
          }
```

```
55                  }
56              ]
57          }
58      ]
59  }
```

Código Fonte A.14: Exemplo de código JSON retornado pela URI **`/nobi/controller/create-qos-path`** em uma rede SDN controlada pelo *OpenDaylight Beryllium-SR4*

Com base no que foi apresentado nas funções anteriores, o leitor poderá perceber que um diferencial entre NoBI e as abordagens descritas na Tabela 3.1 (pág. 50) do Capítulo 3 é que, em NoBI, são definidos 2 (dois) modelos de dados unificados: um para a identificação da regra a ser removida com a URI **`/nobi/controller/delete-flow`** e um outro para representação de regra para reserva de largura de banda usada na URI **`/nobi/controller/create-qos-path`**. Estes modelos de dados, juntamente com as demais URIs NoBI, garantem a plena portabilidade das operações realizadas, permitindo ao gerente trabalhar com um conjunto padronizado de *tags*, em vez de lidar com detalhes complexos de representação de instruções dos controladores *Floodlight* e *OpenDaylight*.

Apêndice B

Questionário de Avaliação

O corrente anexo apresenta o questionário utilizado no processo de coleta de informações dos participantes durante a avaliação qualitativa da abordagem NoBI.

Comparação de APIs para Acomodação de Instruções em Redes Definidas por Software

Prezado Participante,

este questionário foi criado para avaliar diferentes APIs voltadas para instalação de instruções em Redes Definidas por Software (SDN). O objetivo é avaliar a facilidade geral de uso das funções de cada API.

Este questionário é parte integrante de um estudo em Redes Definidas por Software desenvolvido na Universidade Federal de Campina Grande (UFCG) pelo estudante de doutorado Cesar Vasconcelos (cesarvasconcelos@copin.ufcg.edu.br), com orientação dos professores Dr. Reinaldo Gomes (reinaldo@computacao.ufcg.edu.br) e Dr. Anderson Costa (afbfc@cin.ufpe.br). Sua participação é de extrema importância para nosso estudo. Garantimos o mais rigoroso sigilo das informações aqui declaradas, através da omissão total de quaisquer informações que permitam sua identificação.

Serão apresentadas a você algumas invocações à URIs, acessadas via requisição HTTP, feitas pelo gerente de redes para instalar instruções ou consultar informações da rede.

Desde já, muito obrigado por sua colaboração!
Cesar Vasconcelos - cesarvasconcelos@copin.ufcg.edu.br

O tempo estimado para responder o questionário é de apenas 5 min.

*Obrigatório

Informações Iniciais

Todas informações a seguir são sigilosas e não serão publicadas.

1. **Nome** *

2. **Endereço de e-mail** *

3. **Informe o setor em que se enquadra a empresa que trabalha atualmente** *
   Marque todas que se aplicam.
   - [ ] Privado.
   - [ ] Público.
   - [ ] Acadêmico.
   - [ ] Outro: _____

4. **Experiência com a área de Redes Definidas por Software** *
   Marcar apenas uma oval.
   - ◯ Tenho utilizado redes SDN em projetos pessoais.
   - ◯ Tenho experiência acadêmica com redes SDN (e.g., projetos de pesquisa).
   - ◯ Tenho experiência em redes SDN no ambiente coorporativo.
   - ◯ Outro: _____

5. **Tempo de experiência em Redes Definidas por Software** *
   Marcar apenas uma oval.
   - ◯ Menos de 1 ano.
   - ◯ Entre 1 e 3 anos.
   - ◯ Entre 4 e 6 anos.
   - ◯ Mais de 6 anos.
   - ◯ Outro: _____

Utilização das APIs

Para todas as questões abaixo, considere que o gerente de redes irá acessar URIs de diferentes APIs Web por meio de envio de mensagens HTTP com o intuito de comunicar-se com o controlador existente para consultar informações da rede SDN.

6. **Considerando que o gerente de redes deseja consultar informações quanto à topologia SDN, qual das URIs abaixo é mais intuitiva e simples de usar?** *
   Marcar apenas uma oval.
   - ◯ GET /nobi/controller/topology
   - ◯ GET /wm/core/controller/summary/json
   - ◯ GET /json/restconf/operational/network-topology:network-topology/
   - ◯ Os formatos das URIs das APIs possuem a mesma simplicidade e mesma capacidade de transmitir a funcionalidade a ser executada.

7. **Considerando que o gerente de redes deseja obter uma listagem com as regras de controle de fluxo de pacotes instaladas e ativas no equipamento, qual das URIs abaixo é mais intuitiva e simples de usar?** *
   * O termo <switch-id> no formato da URI representa o identificador do equipamento a ser consultado e deve ser fornecido pelo gerente, quando for o caso.
   Marcar apenas uma oval.

   - ◯ GET/restconf/operational/opendaylight-inventory:nodes/node/<switch-id>/table/0
   - ◯ GET /nobi/controller/active-flows
   - ◯ GET /wm/staticflowpusher/list/<switch-id>/json
   - ◯ Os formatos das URIs das APIs possuem a mesma simplicidade e mesma capacidade de transmitir a funcionalidade a ser executada.

8. **Considerando que o gerente de redes deseja obter informações quanto aos switches da rede, qual das URIs abaixo é mais intuitiva e simples de usar?** *
   Marcar apenas uma oval.

   - ◯ GET /nobi/controller/switches
   - ◯ GET/restconf/operational/opendaylight-inventory:nodes/
   - ◯ GET/wm/core/controller/switches/json
   - ◯ Os formatos das URIs das APIs possuem a mesma simplicidade e mesma capacidade de transmitir a funcionalidade a ser executada.

Utilização das APIs

Para as questões abaixo, observe que, enquanto algumas das URIs oferecidas pelas APIs podem ser invocadas pelo gerente diretamente, outras podem exigir que instruções (comandos) sejam passadas como argumento via arquivo no formato JSON ou XML no momento da requisição HTTP.

9. **Considerando que o gerente de redes deseja instalar uma instrução na rede para que pacotes sejam enviados ao controlador, qual das URIs é mais intuitiva e simples de usar?** *
   * Nas figuras abaixo, as URIs oferecidas pelas APIs ao gerente são destacadas na primeira linha, enquanto que o arquivo de comandos a ser repassado pelo gerente (quando for o caso) é mostrado nas linhas seguintes.
   Marcar apenas uma oval.

   - ◯ Figura 1 mostra a URI com acesso mais simples de usar.
   - ◯ Figura 2 mostra a URI com acesso mais simples de usar.
   - ◯ Figura 3 mostra a URI com acesso mais simples de usar.
   - ◯ Todas as URIs das APIs mostradas nas figuras possuem a mesma simplicidade de uso.

Figura 1:

```
PUT /restconf/config/opendaylight-inventory:nodes/node/<switch-id>/table/0/flow/260

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<flow xmlns="urn:opendaylight:flow:inventory">
    <strict>false</strict>
    <flow-name>FLOW-260-SEND-ALL-TO-CONTROLLER</flow-name>
    <id>260</id>
    <cookie_mask>255</cookie_mask>
    <cookie>105</cookie>
    <table_id>0</table_id>
    <priority>10</priority>
    <hard-timeout>150</hard-timeout>
    <idle-timeout>75</idle-timeout>
    <instructions>
        <instruction>
            <order>0</order>
            <apply-actions>
                <action>
                    <order>0</order>
                    <output-action>
                        <output-node-connector>CONTROLLER</output-node-connector>
                        <max-length>65535</max-length>
                    </output-action>
                </action>
            </apply-actions>
        </instruction>
    </instructions>
</flow>
```

Figura 2:

```
POST /wm/staticflowpusher/json

{
  "switch":"00:01:cc:3e:5f:81:ea:01",
  "name":"FLOW-260-SEND-ALL-TO-CONTROLLER",
  "cookie":"105",
  "priority": "10",
  "idleTimeoutSec": "75",
  "hardTimeoutSec": "150",
  "flags": "0",
  "active":"true",
  "actions": "output=controller"
}
```

Figura 3:

```
POST /nobi/controller/send-to-controller

// nesta API, o arquivo com os comandos é produzido automaticamente
// e não precisa ser passado pelo gerente como argumento na requisição
```

Utilização das APIs

10. **Considerando que o gerente de redes deseja instalar uma instrução na rede para encaminhar tráfego UDP marcado com DSCP 34 (AF 41) a uma fila de serviço (Q4) para a reserva de largura de banda mínima (12Mbps) na comunicação entre dois pontos da rede com endereços IP de origem 192.168.24.10 e destino 192.168.24.24, qual das URIs é mais intuitiva e simples de usar?** *

* Nas figuras abaixo, as URIs oferecidas pelas APIs ao gerente são destacadas na primeira linha e o arquivo de comandos que deve ser repassado pelo gerente (quando for o caso) é mostrado nas linhas seguintes.
Marcar apenas uma oval.

- ◯ Figura 1 mostra a URI com acesso mais simples de usar.
- ◯ Figura 2 mostra a URI com acesso mais simples de usar.
- ◯ Figura 3 mostra a URI com acesso mais simples de usar.
- ◯ Todas as URIs das APIs mostradas nas figuras possuem a mesma simplicidade de uso.

Figura 1:

```
PUT /restconf/config/opendaylight-inventory:nodes/node/<switch-id>/table/0/flow/99

<flow xmlns="urn:opendaylight:flow:inventory">
  <id>99</id>
  <flow-name>APPLY-Q4-UDP-HOST1-HOST2</flow-name>
    <flags>
    SEND_FLOW_REM
    </flags>
  <instructions>
    <instruction>
    <order>0</order>
    <apply-actions>
      <action>
        <order>1</order>
        <set-queue-action>
          <queue-id>4</queue-id>
          <queue>4</queue>
        </set-queue-action>
      </action>
      <action>
        <order>2</order>
        <output-action>
            <output-node-connector>NORMAL</output-node-connector>
            <max-length>65535</max-length>
        </output-action>
      </action>
    </apply-actions>
    </instruction>
  </instructions>
   <match>
        <ethernet-match>
            <ethernet-type>
                <type>2048</type>
            </ethernet-type>
            <ethernet-source>
                <address>00:00:00:00:00:01</address>
            </ethernet-source>
            <ethernet-destination>
                <address>00:00:00:00:00:02</address>
            </ethernet-destination>
        </ethernet-match>
        <ipv4-source>192.168.24.10/24</ipv4-source>
        <ipv4-destination>192.168.24.24/24</ipv4-destination>
        <ip-match>
            <ip-protocol>17</ip-protocol>
            <ip-dscp>34</ip-dscp>
        </ip-match>
    </match>
  <hard-timeout>0</hard-timeout>
  <idle-timeout>0</idle-timeout>
  <priority>30</priority>
  <table_id>0</table_id>
</flow>
```

Figura 2:

```
POST /wm/staticflowpusher/json

{
  "switch":"00:01:cc:3e:5f:81:59:87",
  "name":"APPLY-Q4-UDP-HOST1-HOST2",
  "cookie":"0",
  "priority":"30",
  "idle_timeout": "0",
  "hard_timeout": "0",
  "active":"true",
  "eth_src": "00:00:00:00:00:01",
  "eth_dst": "00:00:00:00:00:02",
  "eth_type": "2048",
  "ip_proto": "0x11",
  "ip_dscp": "34",
  "ipv4_src": "192.168.24.10/24",
  "ipv4_dst": "192.168.24.24/24",
  "actions":"set_queue=4,output=normal"
}
```

Figura 3:

```
POST /nobi/controller/create-qos-path

{
  "name":"APPLY-Q4-UDP-HOST1-HOST2",
  "eth_type": "2048",
  "ip_proto": "17",
  "ip_dscp": "34",
  "ipv4_src": "192.168.24.10/24",
  "ipv4_dst": "192.168.24.24/24",
  "min_bandwidth_kbps": "12000"
}
```

11. **Em sua opinião, uma API com um corpo de URIs de formato uniforme e ainda capaz de se comunicar de forma transparente com diferentes controladores SDN seria útil para o gerente de redes?** *

Neste caso, esta API deveria traduzir as requisições feitas às suas URIs em dialetos de comandos específicos suportados pela API do controlador existente, de forma automática.
Marcar apenas uma oval.

- ◯ Sim
- ◯ Não