Comparação de Políticas para Configuração de Fluxos em Redes Definidas por *Software*

Paulo Roberto Vieira Jr[1], **Adriano Fiorese**[1], **Guilherme Koslovski**[1]

[1] Universidade do Estado de Santa Catarina (UDESC)
Programa de Pós-Graduação em Computação Aplicada (PPGCA)
Santa Catarina – SC – Brazil

paulo.vieira@ifpr.edu.br, {adriano.fiorese,guilherme.koslovski}@udesc.br

Abstract. *Software Defined Networks (SDN) has disseminated the decoupling between the control plane and the data plane in network technologies. Although promising, this logical separation added complexity in the initial configuration of routing rules. In this context, this article aims to investigate two configuration policies for routing rules on SDN: reactive and proactive. The reactive approach seeks to install the rule on the switch when a packet arrives, while the proactive approach aims to install the rule on all switches involved in the datapath. Experimental results using Floodlight as a controller quantified the time required to configurate a new flow, indicating that the proactive approach is superior in the analyzed scenarios.*

1. Introdução

As redes de computadores são essenciais para empresas, universidades, centros de pesquisa e ambientes governamentais. Vivemos em uma sociedade em que tudo, ou quase tudo, está conectado e uma informação pode ser acessível de praticamente qualquer lugar ou dispositivo. Entretanto, redes tradicionais baseadas no protocolo IP possuem um gerenciamento complexo [Benson et al. 2009] [Kim and Feamster 2013]. Em diversos ambientes, operadores de rede precisam configurar cada dispositivo na rede usando comandos de baixo nível ou frequentemente utilizando *software* embarcado do fabricante, tornando as redes fechadas, proprietárias e verticalmente integradas. Nestas redes é difícil experimentar novas ideias, testar novos protocolos, inovar ou fazer melhorias.

Redes Definidas por Software (*Software Defined Networking* - SDN) é um paradigma que surgiu em resposta às limitações da infraestrutura física de redes tradicionais, quebrando a integração vertical e separando a camada de dados da camada de controle, permitindo que novas ideias e protocolos sejam experimentados sem prejudicar o tráfego de produção [Jain and Paul 2013] [Kreutz et al. 2015]. Nesse cenário, roteadores e *swtiches* tornam-se dispositivos que apenas encaminham o tráfego baseado nas decisões tomadas por um controlador (*controller*), elemento que assume o papel da camada de controle. A comunicação entre o controlador e os dispositivos de rede pode ser feita através de protocolos específicos, como por exemplo o *OpenFlow* [McKeown et al. 2008].

Em SDNs, um controlador (centralizado ou distribuído) é a entidade que gerencia todos os dispositivos da rede, adicionando ou removendo fluxos nas tabelas dos *switches* [Kreutz et al. 2015].

Um exemplo de controlador centralizado é o projeto *Floodlight* [Floodlight 2012], implementado em *Java*, que oferece mecanismos para interação com *switches* físicos e virtuais através do protocolo *OpenFlow*.

É sabido que a comunicação entre controladores, centralizados ou distribuídos, e *switches* acrescenta latência no encaminhamento de pacotes [He et al. 2015]. Nesse contexto, este artigo quantifica o tempo de instalação de um fluxo em *switches* com suporte *OpenFlow*, gerenciados pelo controlador *Floodlight*, usando duas abordagens: reativa e pró-ativa.

O restante do trabalho está organizado da seguinte forma: A Seção 2 revisa as definições de SDN, enquanto a Seção 3 descreve o cenário e a topologia utilizada no experimento. Por sua, vez, a Seção 4 descreve as políticas reativa e pró-ativa. A análise experimental é discutida na Seção 5 enquanto a conclusão e trabalhos futuros são apresentados na seção 6.

2. Redes Definidas por Sofware

Em redes IPs tradicionais, roteadores e *switches* fazem o papel da camada de controle e da camada de dados. Estas duas camadas são embutidas nos dispositivos de rede, reduzindo a flexibilidade, inovação e evolução da infraestrutura. Sobretudo, geralmente a configuração destes dispositivos deve ser feita manualmente através de um *software* do fabricante ou utilizando uma linguagem específica.

Em suma, a camada de controle decide o que fazer com os pacotes, sendo responsável pelo gerenciamento do tráfego da rede, enquanto a camada de dados apenas encaminha o tráfego de acordo com as decisões tomadas pela camada de controle [Kreutz et al. 2015].

Redes Definidas por *Software* (*Software Defined Networking* - SDN) é um paradigma de rede emergente que separa a camada de controle da camada de dados, prometendo melhorar a utilização dos recursos de rede, simplificar o gerenciamento de rede, reduzir custos de operação e promover inovação e evolução [Akyildiz et al. 2014]. Em uma arquitetura SDN (exemplificada na Figura 1) o nível mais baixo representa a infraestrutura de rede ou camada de dados, ou seja, nessa camada os dispositivos apenas encaminham o tráfego baseado nas decisões da camada de controle.

A comunicação entre a camada de dados e a camada de controle pode ser feita utilizando protocolos como o *OpenFlow*, *SNMP*, *BGP*, *NetConf* entre outros, porém muitos controladores suportam somente o protocolo *OpenFlow*.

Ainda na Figura 1, o nível acima da camada de dados é a camada de controle. O elemento chave dessa camada é o controlador, responsável por gerenciar todos os dispositivos da rede, além de adicionar ou remover fluxos das tabelas de fluxos nos dispositivos de rede.

O controlador pode ser centralizado ou distribuído. Um controlador centralizado pode ser um ponto de falha na rede e pode possuir limitações de escalabilidade, além de introduzir um atraso no encaminhamento de pacote causado pela interação entre a camada de controle e a camada de dados [Jain and Paul 2013], já um controlador distribuído não possui essas limitações, porém pode possuir problemas comuns à sistemas distribuídos como por exemplo problemas na sincronia do estado na rede. No último nível estão as

aplicações de rede. As aplicações são desenvolvidas utilizando uma *Northbound API (Application Programming Interface)* fornecida pelo desenvolvedor do controlador. Estas APIs fornecem um nível de abstração que permite às aplicações não depender de detalhes específicos da infraestrutura. Como exemplos de aplicações pode-se citar: protocolos de roteamento, balanceamento de carga, monitoração, detecção de ataques, aplicação de políticas.

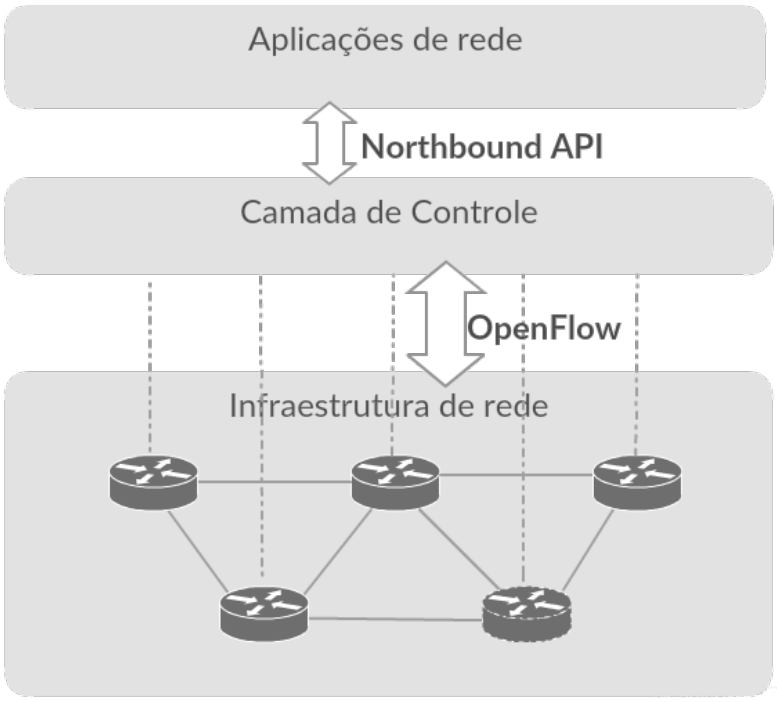

Figura 1. Arquitetura de uma Rede Definida por *Software*.

O protocolo *OpenFlow*, foco do presente trabalho, é um protocolo aberto para gerenciar a tabela de fluxos em diferentes *switches* e roteadores, fornecendo uma forma padronizada para um controlador se comunicar com um dispositivo de rede. Atualmente, o protocolo é mantido pelo *The OpenFlow Consortium*, que tem como objetivo popularizar o *OpenFlow* e manter a especificação de um *switch OpenFlow*. Com o *OpenFlow* é possível separar o tráfego de rede em fluxos de produção e fluxos de pesquisa, permitindo a pesquisadores controlar seus próprios fluxos, testar novos protocolos, implementar novos modelos de segurança e até mesmo alternativas ao IP.

Um fluxo é uma sequência de pacotes que corresponde a uma entrada específica numa tabela de fluxo. Uma combinação de entradas de fluxo de múltiplos *switches* definem um fluxo que está vinculado a um caminho específico. Em *OpenFlow*, os *switches* podem ser classificados em dois tipos: *OpenFlow only* e *OpenFlow hybrid*. O *switch OpenFlow only* funciona somente com o protocolo *OpenFlow*, encaminhando pacotes entre portas, seguindo a definição do controlador. Já o *switch OpenFlow hybrid* atua como um dispositivo normal de rede, além de suportar o protocolo *OpenFlow*.

A Figura 2 apresenta a arquitetura de um *switch OpenFlow* de acordo com a especificação do protocolo *OpenFlow* (em sua versão 1.5), e deve possuir os seguintes componentes: uma ou mais tabelas de fluxos, uma tabela de grupo, um canal de

comunicação para um controlador externo utilizando o protocolo *OpenFlow* ou outro protocolo como *SNMP, BPG ou NetConf*, uma tabela de métricas e um ou mais controladores externos.

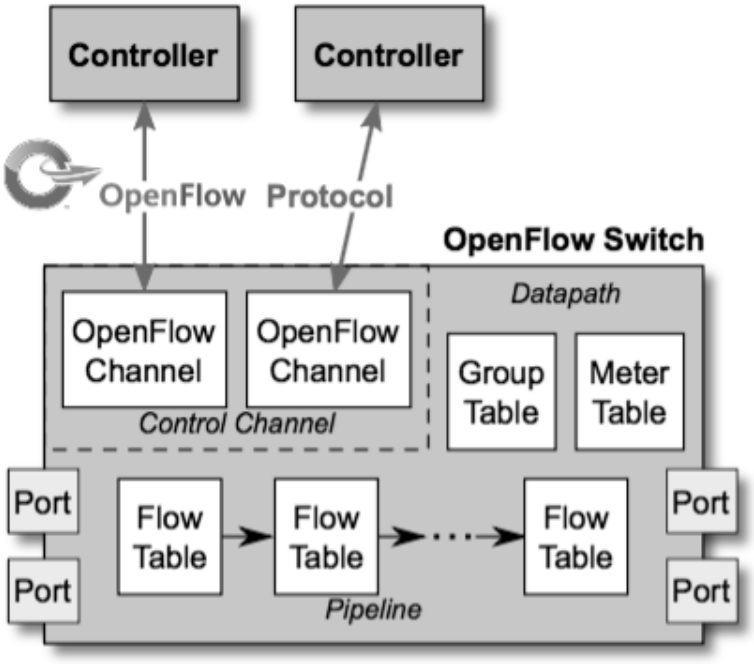

Figura 2. Arquitetura de um switch OpenFlow.
Fonte: [Foundation 2015]

Com SDN é possível implementar aplicações de controle de rede que se adaptam às mudanças na tolopogia de rede, padrões de tráfego ou horário do dia. Redes muito grandes com constantes mudanças precisam configurar a camada de dados com maior frequência sem interromper serviços e sem afetar clientes. O tempo de instalação dos fluxos nos *switches* é importante na escolha de uma política para configuração de fluxos sob demanda (à medida em que novos pacotes chegam). Uma abordagem com menor desempenho pode impedir o acesso à serviços ou o tornar lento, afetar clientes ou até mesmo causar perdas financeiras.

3. Metodologia e Trabalhos Relacionados

Dentre os controladores existentes, selecionamos para a realização deste trabalho o controlador *Floodlight* por apresentar um estágio maduro de desenvolvimento e ser adotado em diversos cenários experimentais e industriais.

Duas abordagens foram utilizadas para medir o tempo de instalação de um fluxo: reativa e pró-ativa. Na abordagem reativa a instalação dos fluxos ocorre no momento em que os pacotes são vistos em cada salto na rede, já na abordagem pró-ativa a instalação dos fluxos pode ocorrer antes da chegada do pacote.

Na literatura encontramos algumas referências para estas abordagens. Por exemplo, [Bifulco and Matsiuk 2015] utiliza somente a abordagem reativa e realiza a medição de tempo de instalação de 10K de fluxos em um *hardware* comercial. Por sua

vez, [Tootoonchian et al. 2012] desenvolveram uma ferramenta para quantificar o desempenho de um controlador medindo o número de instalações de fluxos por segundo que um controlador pode manipular. A ferramenta emula o comportamento de um *switch OpenFlow* enviando mensagens $\{packet\_in\}$ para o controlador. [Dusi et al. 2014] investiga os requisitos de tamanho das tabelas de fluxos e como o tempo limite (*timeout*) dos fluxos afeta as tabelas utilizando as duas abordagens.

De forma complementar, este trabalho tem como objetivo investigar o tempo de instalação de um fluxo em um *switch OpenFlow* usando as abordagens reativa e pró-ativa, comparando os tempos entre elas. Para isso, um cliente UDP registra o tempo de saída de um pacote até um servidor UDP e o tempo em que a resposta foi recebida, e o resultado é a diferença entre esses tempos. Foram implementados dois módulos no *Floodlight* para a instalação dos fluxos usando as duas abordagens e o *Floodlight* foi executado com o módulo de encaminhamento padrão (a saber, *forwarding*) desabilitado.

4. Políticas para Configuração de Fluxos em *Switches* SDN

A habilidade de configurar ou reconfigurar o comportamento das redes permite a criação de aplicações que reagem ao estado da rede. Esta reação deve ser rápida e o processo deve ocorrer no menor tempo possível, evitando uma indisponibilidade de serviço temporária. Em uma rede SDN, o processo de configuração é realizado através na manipulação dos fluxos, adicionando-os ou removendo-os das tabelas de fluxos nos *switches*. Neste trabalho, duas abordagens para a instalação de fluxos são descritas a seguir: reativa e pró-ativa.

4.1. Abordagem Reativa

Na abordagem reativa, apresentada na Figura 3, os *switches* que compõem o caminho físico de um fluxo são configurados individualmente sempre que um novo pacote é encaminhado ao controlador. Exemplificando, a rede se comporta da seguinte forma: um cliente UDP faz uma requisição para um servidor UDP; quando o pacote da requisição chega ao primeiro *switch*, este consulta o controlador sobre o que deve ser feito com o pacote. O controlador responde instalando um fluxo no *switch* fazendo com que o pacote seja encaminhado para o próximo *switch* do caminho. Quando o pacote chega ao próximo *switch*, o processo se repete até que o pacote chegue ao servidor. O cliente, por sua vez, registra o tempo total desde a saída da requisição até o momento em que recebe a resposta do servidor. Nesta abordagem, cada *switch* consulta o controlador sempre que um novo pacote chega e a instalação do fluxo é feita a cada salto do pacote.

4.2. Abordagem Pró-Ativa

Na abordagem pró-ativa, apresentada na Figura 4, o controlador, ao receber a primeira solicitação de configuração enviada pelo *switch* de borda da comunicação, realizar a configuração do caminho completo.

Essa abordagem pode ser exemplificada da seguinte forma: um cliente UDP faz uma requisição para um servidor UDP, quando o pacote da requisição chega ao primeiro *switch*, este consulta o controlador sobre o que deve ser feito com o pacote. O controlador tem conhecimento da topologia da rede fazendo a instalação de um fluxo diretamente em todos os *switches* envolvidos no caminho até o servidor. Assim, quando o pacote chega ao próximo *switch* o fluxo já está instalado, não necessitando uma nova consulta do

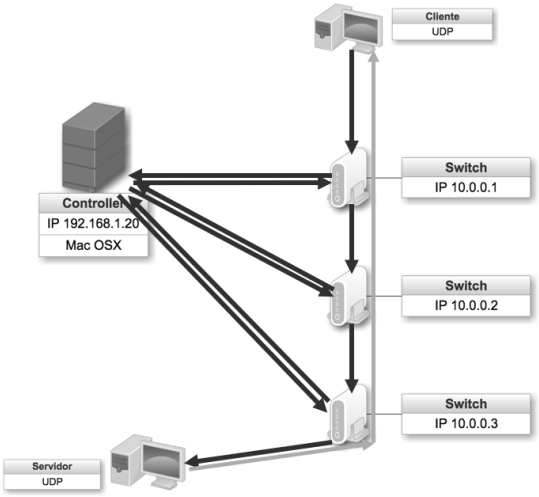

Figura 3. Abordagem Reativa.

controlador. Essa abordagem evita a latência de uma nova consulta. Novamente, o cliente registra o tempo total desde a saída da requisição até o momento em que recebe a resposta do servidor.

De acordo com [Tootoonchian et al. 2012], a abordagem pró-ativa pode instalar os fluxos estaticamente antes da chegada dos pacotes. Neste trabalho, no entanto, a instalação dos fluxos ocorre no momento em que o pacote é visto pela primeira vez na rede, ou seja, quando o primeiro pacote enviado da mensagem atinge o primeiro *switch* na topologia da rede.

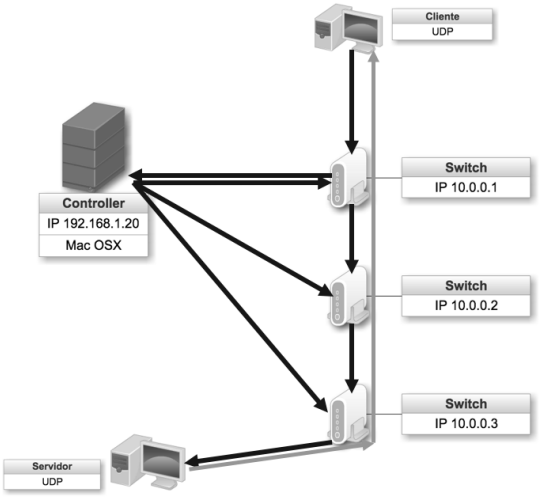

Figura 4. Abordagem pró-ativa.

4.3. Implementação das Políticas de Configuração

O *Floodlight* é um controlador centralizado escrito em linguagem *Java*, que suporta *switches* físicos e virtuais, baseados no protocolo *OpenFlow*. Esse controlador possui, entre

outros recursos, um sistema de módulos que simplifica o desenvolvimento e utilização de seu mecanismo, além de fornecer uma *Application Programming Interface* (API) em linguagem *Java* para o desenvolvimento de aplicações e módulos [Floodlight 2012].

Para implementar as abordagens discutidas, dois módulos foram implementados, e o módulo padrão de encaminhamento e configuração de fluxos (*forwarding*) do *Floodlight* foi desabilitado a fim de obter-se maior controle dos fluxos, instalando na rede somente os fluxos necessários para o experimento. O módulo *forwarding* permite ao *Floodlight* operar em redes que contenham *switches OpenFlow* e não *OpenFlow* encaminhando pacotes entre dois dispositivos.

Os fluxos instalados nas duas abordagens são criados com o IP de origem e destino configurados, não possuem *hardtimeout* e nem *idletimeout*. Desta forma o fluxo não é removido do *switch* após um limite de tempo, e sim somente após um comando do controlador ou quando o *switch* for reiniciado.

O Algoritmo 1 apresenta a lógica para a aboradgem reativa e inicia com a chegada de um novo pacote de qualquer *switch* na rede, o algoritmo busca então a origem do pacote e o destino, criando uma nova entrada para a tabela de fluxos com estas informações. O próximo passo é a instalação da entrada somente no *switch* que recebeu o evento de novo pacote para em seguida encaminhar o pacote para o próximo *switch* do caminho. Já o Algoritmo 2, referente a abordagem pró-ativa, difere do Algoritmo 1 pela utilização do serviço de descoberta de *links* do *Floodlight* que busca um caminho entre a origem e o destino e retorna os *switches* entre esses dois pontos. A instalação da entrada de fluxo é realizada em todos os *switches* retornados pela serviço de descoberta de *links*.

Algoritmo 1 Algoritmo Reativo

1: $pkt = receiveFromSwitch(switch)$;
2: $src = getSource(pkt)$;
3: $dst = getDestination(pkt)$;
4: $flowEntry = createNewFlowEntry(src, dst)$;
5: $installNewFlowEntry(switch, flowEntry)$; **return**

Algoritmo 2 Algoritmo Pró-ativo

1: $pkt = receiveFromSwitch(switch)$;
2: $src = getSource(pkt)$;
3: $dst = getDestination(pkt)$;
4: $flowEntry = createNewFlowEntry(src, dst)$;
5: $topology = getNetworkTopology()$;
6: $path = topology.findPath(src, dst)$
7: **for** $i = 0$ **to** path.size()
8: $\quad installNewFlowEntry(path.getSwitch(i), flowEntry)$;
return

Para quantificação do tempo de configuração, o cliente e o servidor UDP foram implementados em *Java*. O cliente faz o envio de um datagrama com 512 *bytes* de tamanho máximo e informa a diferença entre o tempo de saída do datagrama e o recebimento da resposta do servidor em milissegundos. O servidor faz apenas um *echo* para o cliente.

5. Análise Experimental

Para este trabalho foram utilizados os seguintes recursos: MacOS 10.10, Java JDK 1.7, Floodlight 1.0, Virtual Box 4.3.30 com uma máquina virtual Ubuntu 14.04 LTS e Mininet 2.2.1.

Quinze cenários foram utilizados na análise experimental. O cenário base foi desenhado da seguinte maneira: o *Floodlight* foi executado no Mac OS. O Ubuntu 14.04 foi executado no Virtual Box e nele foi criado um *script* em *Python* para gerar a topologia de rede no Mininet.

O Mininet é um sistema de prototipagem rápida de grandes redes para computadores com recursos limitados. Possui uma abordagem leve de utilização dos recursos de virtualização em nível de sistema operacional, incluindo processos e *network namespaces* permitindo escalar centenas de nós [Lantz et al. 2010].

A topologia criada no Mininet possui dois *hosts* virtuais com papéis de cliente UDP e servidor UDP e *switches* virtuais configurados para se comunicar com o *Floodlight* executado no MacOS. A Figura 5 mostra um exemplo de topologia com dois *switches*, um cliente UDP, um servidor UDP e o controlador executado no MacOS. Uma topologia em linha foi utilizada em todos os cenários.

O primeiro cenário possui na topologia apenas um *switch*, o cliente UDP e o servidor UDP estão conectados neste único *switch*. O segundo cenário possui dois *switches*. O cliente UDP e o servidor UDP ficam nas pontas da topologia e entre eles, os *switches* conectados em uma topologia linear. Os cenários se diferenciam um do outro apenas na quantidade de *switches* na topologia, a cada cenário experimentado a quantidade de *switches* foi aumentada. Antes de cada execução um comando para "limpeza" da *Mininet* era executado para em seguida ser executado o *script* de criação da topologia. Os cenários foram experimentados com as seguintes quantidades de *switches*: 1, 2, 3, 4, 5, 10, 20, 30, 40 50, 60, 70, 80, 90 e 100.

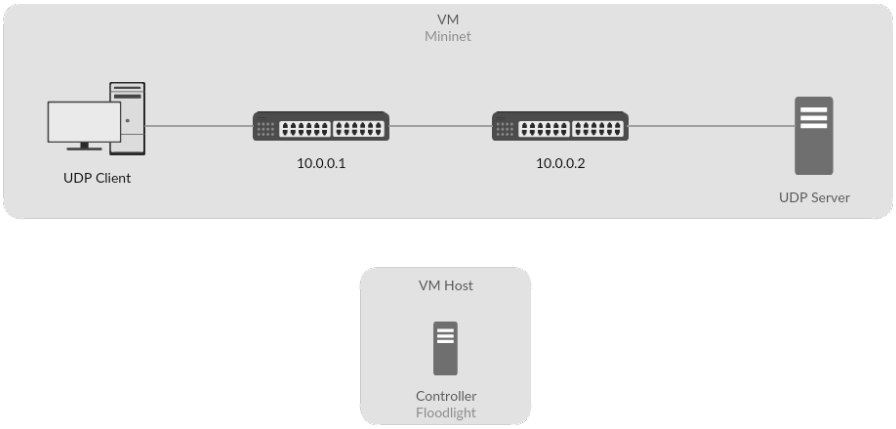

Figura 5. Exemplo de topologia.

Cada cenário foi executado cinco vezes em cada abordagem. Uma média aritmética foi utilizada para se obter o tempo médio da instalação dos fluxos de cada cenário. A Figura 6 apresenta o gráfico dos resultados obtidos. O eixo *x* apresenta a

quantidade de *switches* na topologia em cada cenário. O eixo *y* apresenta o tempo em milissegundos. Percebe-se que à medida que a quantidade de *switches* na topologia aumenta, a abordagem reativa leva mais tempo para instalar os fluxos, pois a cada salto do pacote na rede, o *switch* precisa consultar o controlador aumentando o tempo total da resposta para o cliente.

A abordagem pró-ativa obteve valores menores na maioria dos casos, mesmo aumentando a quantidade de *switches* na topologia. As duas abordagens obtiveram tempo de instalação de um fluxo muito próximos até 20 *switches* na topologia. O desempenho da abordagem pró-ativa deve-se ao fato de que a cada salto do pacote na rede não há necessidade da consulta ao controlador em cada *switch*. O tempo médio para instalação de um fluxo em 100 *switches* na abordagem reativa foi de 2783ms enquanto que na abordagem pró-ativa o tempo para 100 *switches* foi de 1263ms.

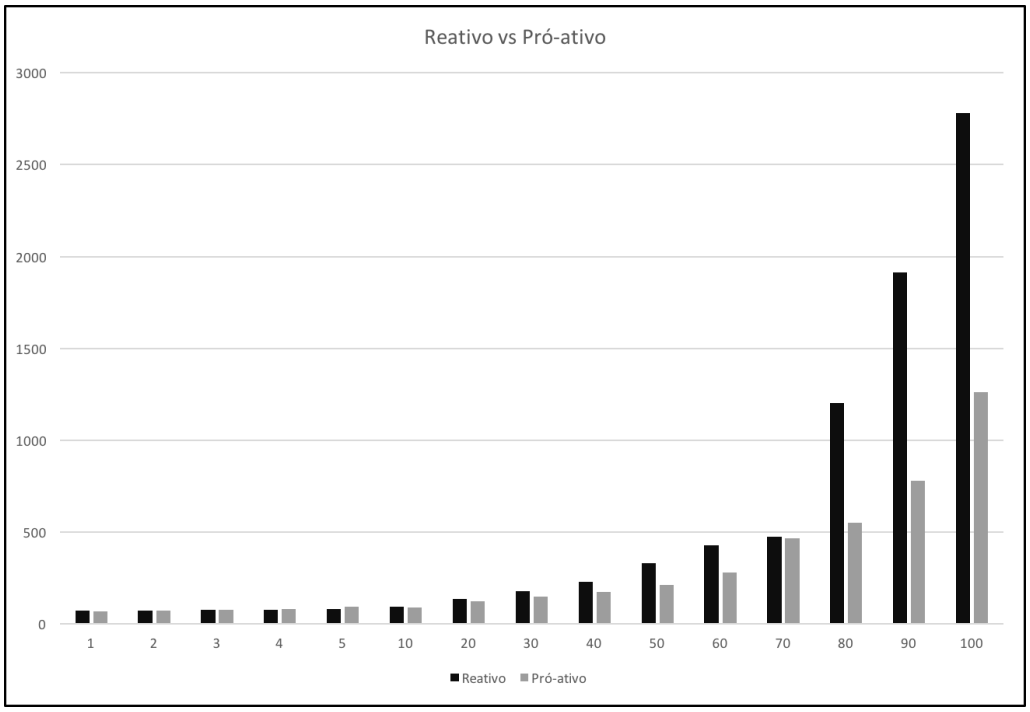

Figura 6. Resultados experimentais.

6. Conclusão

Este trabalho apresentou uma investigação do tempo de instalação de um fluxo UDP em uma topologia linear, variando a quantidade de *switches* de um a cem. Duas abordagens foram utilizadas para configuração dos fluxos nos *switches*: abordagem reativa e pró-ativa. A abordagem pró-ativa mostrou-se a opção com melhor desempenho nos experimentos realizados, obtendo menores tempos de instalação de um fluxo no cenário estudado. Como trabalhos futuros, pretende-se, inicialmente, realizar uma quantificação com *hardware* com suporte a *OpenFlow*, bem como utilizar estas abordagens para dimensionar a quantidade de controladores necessários dada uma topologia de rede, além de experimentar as abordagens apresentadas em outras topologias de redes.

Referências

Akyildiz, I. F., Lee, A., Wang, P., Luo, M., and Chou, W. (2014). A roadmap for traffic engineering in sdn-openflow networks. *Computer Networks*, 71:1–30.

Benson, T., Akella, A., and Maltz, D. A. (2009). Unraveling the complexity of network management. In *NSDI*, pages 335–348.

Bifulco, R. and Matsiuk, A. (2015). Towards scalable sdn switches: Enabling faster flow table entries installation. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 343–344. ACM.

Dusi, M., Bifulco, R., Gringoli, F., and Schneider, F. (2014). Reactive logic in software-defined networking: Measuring flow-table requirements. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2014 International*, pages 340–345. IEEE.

Floodlight (2012). Floodlight is an open sdn controller. `http://www.projectfloodlight.org/floodlight`. Acessado em: 2015-10-27.

Foundation, O. N. (2015). Openflow switch specification v1.5.1. `https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf`. Acessado em: 2015-10-30.

He, K., Khalid, J., Gember-Jacobson, A., Das, S., Prakash, C., Akella, A., Li, L. E., and Thottan, M. (2015). Measuring control plane latency in sdn-enabled switches. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, page 25. ACM.

Jain, R. and Paul, S. (2013). Network virtualization and software defined networking for cloud computing: a survey. *Communications Magazine, IEEE*, 51(11):24–31.

Kim, H. and Feamster, N. (2013). Improving network management with software defined networking. *Communications Magazine, IEEE*, 51(2):114–119.

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

Lantz, B., Heller, B., and McKeown, N. (2010). A network in a laptop: Rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pages 19:1–19:6, New York, NY, USA. ACM.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74.

Tootoonchian, A., Gorbunov, S., Ganjali, Y., Casado, M., and Sherwood, R. (2012). On controller performance in software-defined networks. In *USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, volume 54.