Uma estratégia para o serviço de cálculo de caminhos em redes definidas por software

Davison Holanda Pacheco[1]
Airton Nobumasa Ishimori[1]
Fernando Nazareno Nascimento Farias[1]
Antônio Jorge Gomes Abelém[1]

Resumo: o paradigma de redes definidas por software (SDN) está sendo investigado como a solução mais promissora para o atual engessamento da internet, uma vez que propõe a dissociação entre o plano de dados e o plano de controle, proporcionando maior programabilidade às redes de computadores. No entanto, ainda há lacunas em serviços disponíveis nessa arquitetura, dentre as quais se observa o serviço de cálculos de caminhos, que não está evoluindo consideravelmente entre os controladores. Por exemplo, a reserva de recursos, a partir dos requisitos necessários de cada aplicação, permanece como um desafio a ser vencido. Este artigo apresenta uma estratégia de cálculo de caminhos para redes SDN. O objetivo é oferecer um serviço mais flexível no estabelecimento de fluxos OpenFlow, além de possibilitar restrições determinísticas de qualidade vindas das aplicações. A proposta contribui também com uma arquitetura que pode ser aplicada a controladores SDN, um algoritmo de busca, baseado em uma métrica de qualidade de serviço (QoS), e uma análise de desempenho, mostrando que o algoritmo é capaz de minimizar o tempo de busca, processamento e consumo de memória pelo controlador na rede SDN.

Palavras-chave: Algoritmo. OpenFlow. Redes definidas por software.

Abstract: *The paradigm of Software Defined Networking (SDN) has been investigated as the most promising solution to the current inflexibility of the Internet, proposing decoupling between the data plane and the control plane, providing greater programmability to networks. However, there are still gaps in services available in this architecture, among which is observed the service path computation, which has not evolved enough between controllers. For example, the resource reservation from the requirements of each application is still a challenge to be overcome. This paper presents a path computation strategy for SDN networks. The aim is to offer a more flexible service in establishing OpenFlow flows, and enable deterministic constraints welcome applications quality. The proposal also contributes to an architecture that can be applied to SDN controllers, a search algorithm based on a quality of service metric (QoS) and also a performance analysis, which shows the algorithm is able to minimize the search time, processing and memory consumption by the controller in SDN network.*

Keywords: *Algorithm. OpenFlow. Software defined networking.*

1 Introdução

A Internet encontra-se atualmente em uma situação inusitada: o sucesso dessa pode ser considerado estrondoso, já que hoje a tecnologia de interligação de redes de computadores permeia todos os níveis da sociedade. A maioria das atividades da sociedade contemporânea atravessa uma ou mais redes de computadores. No entanto, apesar do sucesso, a arquitetura dessas redes, representada pelos protocolos Transmission Control Protocol/Internet Protocol (TCP/IP), não evoluiu de maneira compatível nos últimos vinte anos. Nesse período, a internet tornou-se comercial e os equipamentos de rede, transformaram-se em caixas-pretas, consideradas

[1] Integrantes do grupo de pesquisa em Redes de Computadores e Comunicação Multimídia, do Instituto de Computação, Universidade Federal do Pará, Belém, Brasil.
{davisonph, airton, fernnf, abelem}@ufpa.br

http://dx.doi.org/10.5335/rbca.v8i2.5392

soluções integradas verticalmente baseadas em software fechado rodando em um hardware proprietário. O resultado desse modelo é o engessamento da internet [1].

Em contraste com a arquitetura da internet atual, novos paradigmas de redes de computadores surgiram objetivando possíveis direcionamentos para o futuro das redes de computadores. Dentre esses paradigmas, pode-se destacar o de redes definidas por softwares (SDN — Software Defined Networking), que segundo [2], trazem duas inovações importantes dentro de sua arquitetura. A primeira está relacionada ao controle das redes, para o qual o paradigma SDN propõe uma ruptura ou divisão dos planos de atuação (entre o plano de controle e o plano de dados) e, dessa maneira, a inteligência da rede (plano de controle) é logicamente centralizada utilizando softwares controladores e seus dispositivos (plano de dados), tornando-se simples encaminhadores ou comutadores de pacotes, que podem ser programados via interface aberta (por ex. Protocolo OpenFlow [3]). A segunda está relacionada ao fortalecimento do conceito de redes programáveis [4], no qual, por meio de interfaces abertas, é possível programar o comportamento do plano de controle de tal modo que reflita em ações aplicadas ao plano de dados. Paralelamente, outro benefício da programabilidade é a permissão da evolução e inovação constante, que permite que esse modelo de rede (SDN) esteja sempre em aperfeiçoamento, de acordo ou não com a demanda de suas aplicações. Finalmente, a programabilidade permite a abstração, facilitando o controle dos tipos de camadas das aplicações (alto ou camada de aplicação, médio ou camada de controle e baixo ou camada de dados ou encaminhamento) [5].

Como característica de uma rede SDN, os controladores são centralizados e têm uma "visão" completa da rede, cuja visão topológica é significativamente útil para aplicações desse paradigma. As topologias podem ser descritas de forma natural e precisa por meio do modelo de grafos, que é um elemento comum em diversos dos principais trabalhos relacionados à área, inclusive os elaborados para SDN [6] e Network Information Base [7].

Com o uso de grafos na representação de topologias, é possível utilizar algoritmos comuns da literatura na busca de caminhos, tais como Dijkstra, buscas em largura, em profundidade, etc. Geralmente, os controladores implementam alguns desses algoritmos ou possuem seus próprios mecanismos de busca, como a aplicação *circuitpusher*[2] pertencente ao controlador floodlight,[3] que é capaz de criar circuitos utilizando recursos de busca de caminhos implementados nesse controlador. Com isso, nota-se a necessidade de formas de busca de caminho relacionadas diretamente ao paradigma SDN.

O objetivo do presente estudo foi descrever uma estratégia de serviço de cálculo de caminhos a ser aplicada em uma arquitetura SDN como suporte durante a tomada de decisão e estabelecimento de fluxos em redes baseadas em OpenFlow, facilitando o uso de aplicações que necessitem do estabelecimento de caminhos ou informações do mapa da rede como apoio para essa decisão.

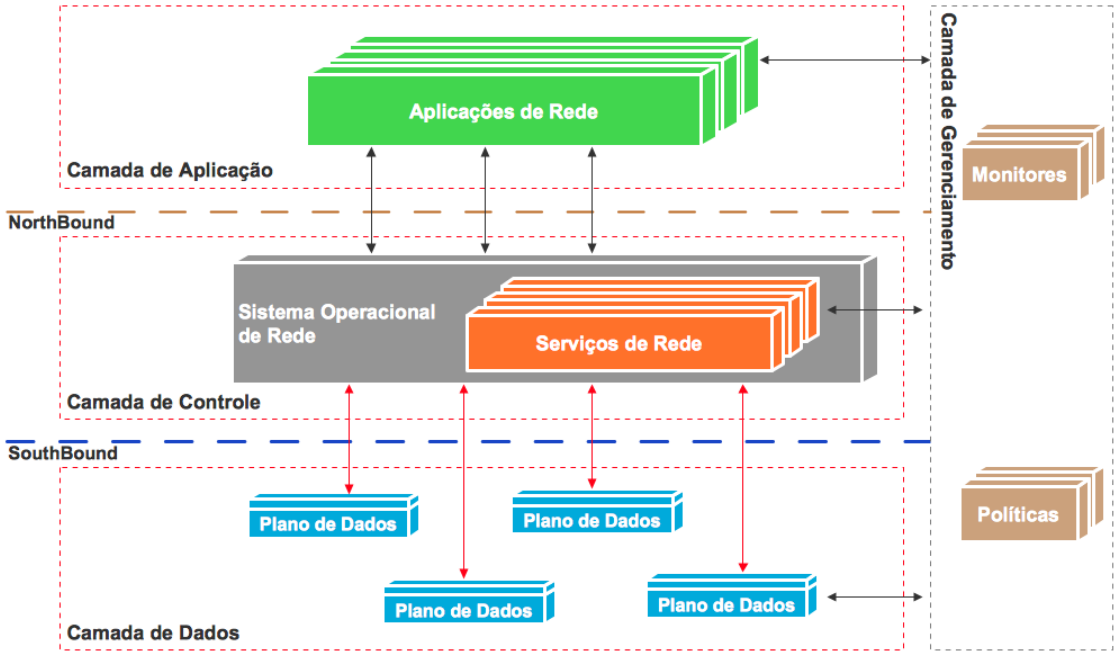
A proposta contribui, pois, com uma arquitetura que possibilita o encontro do melhor caminho dentro de uma rede SDN, baseando-se em requisitos de QoS, além de um algoritmo de busca que apresenta melhor desempenho comparado ao algoritmo de Dijkstra.

2 Fundamentação teórica

Na arquitetura SDN, as camadas de controle e dados são desacopladas e a inteligência e os estados são logicamente centralizados sobre uma infraestrutura de rede que é abstraída pelas aplicações. O objetivo do paradigma SDN é prover interfaces abertas que facilitem o desenvolvimento de aplicações, que sejam capazes de controlar a conectividade promovida por um conjunto de recursos de rede (por ex. switches ou roteadores) e fluxos que por elas atravessam. Adicionalmente, permite a inspeção e modificação da forma ou o comportamento dos tráfegos que estiverem sendo encaminhados pela rede. Como resultado do uso de SDN, há um ganho na programabilidade, automação e controle da rede, que são fatores amplificadores da flexibilidade e rápida evolução da rede, mediante as necessidades de mudança em sua lógica de decisão. Dessa maneira, a arquitetura SDN consiste-se de quatro camadas distintas, acessíveis por meio de APIs e nomeadas da seguinte maneira: camada de aplicação, controle, infraestrutura e gerenciamento (Figura 1).

[2] Disponível em: <http://www.projectfloodlight.org/circuit-pusher/>.
[3] Disponível em: <http://www.projectfloodlight.org/>.

Figura 1: Arquitetura SDN



Fonte: elaborado pelos autores, adaptado de [5].

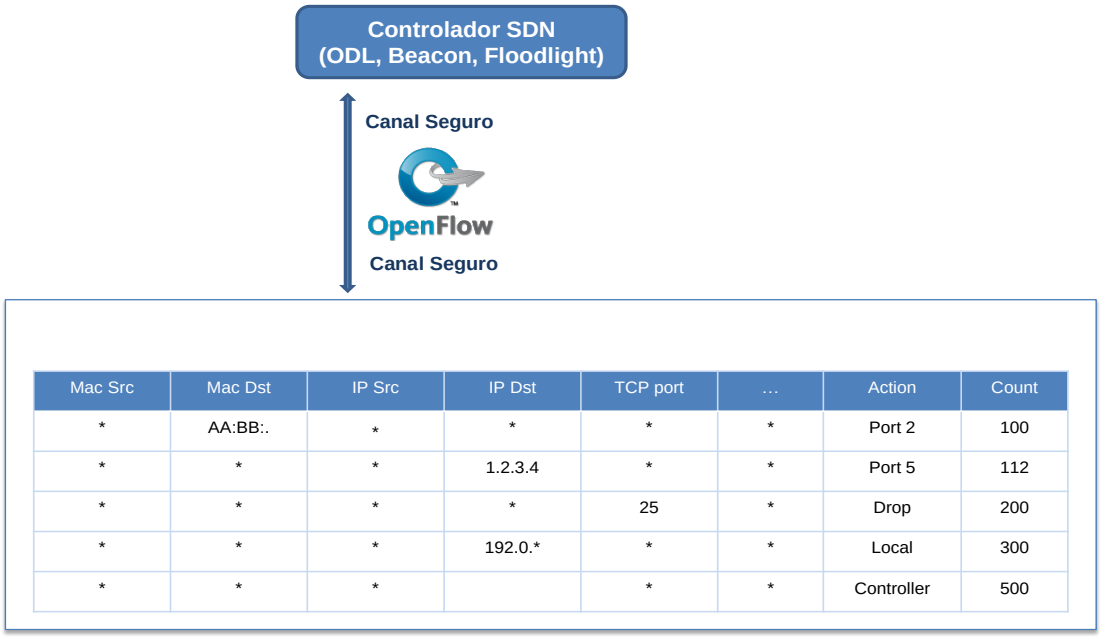
A camada de aplicação consiste nas aplicações dos usuários que consomem um ou mais serviços da camada de controle SDN. Trata-se também de uma fronteira entre aplicação e controle, usualmente conhecida com *northbound*. Já a camada de controle, proporciona sólida funcionalidade de controle que supervisiona o comportamento do encaminhamento dos fluxos de pacotes por meio de interfaces programáticas e abertas, tal como OpenFlow. Por outro lado, a camada de infraestrutura consiste nos elementos de rede, como dispositivos físicos e virtuais que dispõem na comutação e encaminhamento de pacotes. Finalmente, a camada de gerenciamento é responsável por manter o monitoramento sobre o desempenho da rede e o uso de recurso pelos fluxos aplicados no plano de dados. Característica importante, é que essa camada tem a responsabilidade de aplicar as políticas definidas pelos administradores das redes.

2.1 Redes definidas por softwares baseadas em OpenFlow

O OpenFlow foi proposto pela Universidade de Stanford para atender à demanda de validação de novas propostas de arquiteturas e protocolos de rede sobre equipamentos comerciais, tornando-se um protocolo-padrão para determinar as ações de encaminhamento de pacotes em dispositivos de rede, por exemplo, comutadores, roteadores e pontos de acesso sem fio. As regras e ações instaladas no hardware de rede são responsabilidade de um elemento externo, denominado controlador, que pode ser implementado em um servidor comum. Apesar de o OpenFlow ser um dos precursores do conceito SDN, este paradigma não se limitam apenas ao protocolo OpenFlow, podendo ser aplicadas a outros (por ex.: PCEP, ForCES ou NETFCONF/YANG).

De acordo com [8], a forma de gerenciamento baseado em OpenFlow pode ser comparável ao conjunto de instruções de uma CPU, de forma que em OpenFlow essas instruções são armazenadas em abstrações de tabelas de fluxo. Nesse caso, as principais instruções, estabelecidas nas tabelas de fluxo dos dispositivos, são oriundas do controlador OpenFlow ou SDN, conforme ilustra a Figura 2. No entanto, a quantidade de instruções que podem ser manipuladas entre o controlador e o dispositivo dependerá da versão do protocolo, que está compreendida, atualmente, entre as versões 1.0 a 1.4. Essa tupla, ou combinação de campos, abrange informações de cabeçalhos dos protocolos das camadas do TCP/IP, tais como Ethernet, IP, MPLS, TCP ou UDP. Porém, a quantidade de instruções não é limitada e, dependendo do dispositivo, podem ser ampliadas. Vale ressaltar que, tal como se observa na Figura 2, redes baseadas em OpenFlow podem ser divididas entre três principais componentes: dispositivo OpenFlow, canal seguro fundamentado no protocolo OpenFlow e o controlador OpenFlow.

Figura 2: Instruções da tabela de fluxo em um dispositivo OpenFlow



Mac Src	Mac Dst	IP Src	IP Dst	TCP port	…	Action	Count
*	AA:BB:.	*	*	*	*	Port 2	100
*	*	*	1.2.3.4	*	*	Port 5	112
*	*	*	*	25	*	Drop	200
*	*	*	192.0.*	*	*	Local	300
*	*	*		*	*	Controller	500

Fonte: elaborado pelos autores adaptado de [9].

O dispositivo OpenFlow é responsável por encaminhar os fluxos de pacotes baseados nas instruções e ações aplicadas à tabela de fluxo. Além disso, o equipamento também implementa um agente OpenFlow (*softswitch*) que transforma as ações do protocolo em instruções a serem executadas pelo hardware do equipamento e vice-versa, tanto para usuário quanto para *kernel*. Dentre os principais agentes disponíveis, têm-se o OpenvSwitch (OVS) e o CPqD/Ofsoftswitch,[4] dos quais o OVS dá suporte a todas as versões de OpenFlow e o CPqD dá suporte apenas à versão 1.3.

O canal seguro é a ligação entre o controlador e o dispositivo OpenFlow que o controlador utiliza para determinar o tratamento que os fluxos de pacotes sofrerão dentro do equipamento. Essa comunicação é baseada no protocolo OpenFlow, que é um protocolo aberto para comunicação com troca de mensagem por intermédio de um canal de comunicação baseado em TCP ou UDP. As mensagens podem ser simétricas (por ex. *hello*, *echo*, *vendor*), assíncronas (por ex. *packet in*, *flow removed*, *port status*, *error*) ou iniciadas pelo controlador (por ex. *features*, *configuration*, *modifystate*, *sendpacket*, *barrier*). O protocolo é padronizado e mantido pela *Open Networking Foundation*, que também mantém a especificação do protocolo [9].

O controlador OpenFlow/SDN, também denominado de sistema operacional da rede, é a entidade que contém toda a inteligência da rede. Trata-se da camada intermediária entre o plano de dados e o de aplicações. O controlador é composto de uma série de serviços que são consumidos pela camada superior por meio de interfaces como *web-services* ou *restfull*, padronizando a representação desses dados em arquivos xml ou json. Os serviços disponíveis em uma rede OpenFlow variarão de acordo com o controlador, cujos mais conhecidos são: descoberta de topologia, gerenciamento de topologia, gerenciamento de dispositivos conectados, gerenciamento de fluxos aplicados e gerenciamento de mensagens recebidas e enviadas. Dentre os principais controladores abertos disponíveis, têm-se o Floodlight, o OpenDayLight e o ONOS.

O funcionamento da rede OpenFlow pode ser feito de maneira proativa ou reativa. A proativa acontece quando um ou mais serviços do sistema operacional de rede são inicializados, aplicando-se automaticamente instruções prévias aos seus equipamentos e programando imediatamente o comportamento dos fluxos que usarão a rede. O funcionamento reativo é a decisão a ser tomada sobre o comportamento quando o primeiro pacote chega ao equipamento; em seguida esse pacote é enviado ao controlador, que por sua vez analisá-lo-á e decidirá sobre o caminho do fluxo; após a decisão, as instruções são enviadas para os dispositivos que estão relacionados a este caminho.

[4] Projeto *CPqD/Ofsoftswitch*. Disponível em: <https://github.com/CPqD/ofsoftswitch13>.

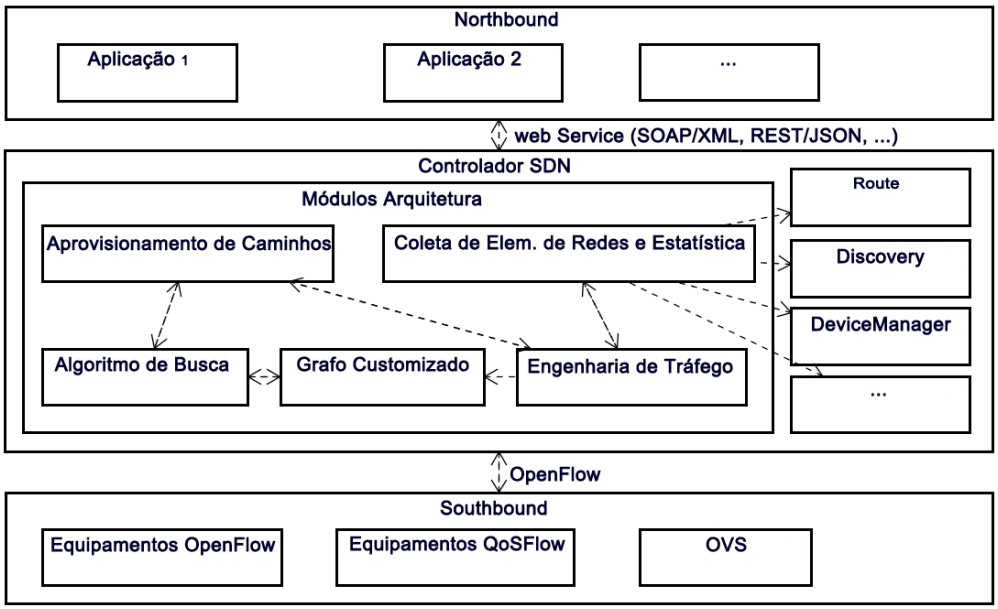
Por conseguinte, a forma como o caminho é calculado não segue um padrão entre os controladores SDN. Na maioria dos casos, implementa-se o algoritmo de menor caminho, que pode ser ruim para o fluxo de dados sensíveis ao contexto da rede (por ex.: atraso, congestionamentos ou variação dos atrasos). O ideal seria um serviço capaz de calcular rotas que levassem em consideração características passadas pela aplicação ou o administrador da rede SDN.

3 Apresentação da proposta

3.1 Arquitetura do serviço de cálculo de caminho

A arquitetura do modelo proposto é baseada na arquitetura SDN de três camadas. A estrutura planejada fundamenta-se na construção de módulos, para obter uma solução flexível e extensível, além de proporcionar melhor organização, maior independência e isolamento entre as funcionalidades providas. A arquitetura prevê a criação e extensão de algumas aplicações da camada de controle, uma vez que sua estrutura está presente no núcleo de um controlador OpenFlow. A arquitetura do serviço proposto está ilustrada na Figura 3.

Figura 3: Arquitetura do serviço de cálculo de caminhos



Fonte: elaborado pelos autores

Analisando-se a arquitetura, na Figura 3 verifica-se a existência de duas extremidades, ideia similar à SDN: a extremidade norte (*northbound*) — que se comunica com aplicações clientes (que podem ser do tipo web ou desktop) — e a extremidade sul (*southbound*) — que se comunica com elementos de comutação da rede.

Na parte correspondente à camada de controle, localiza-se o controlador SDN, ponto em que se localiza toda a inteligência da rede, ou seja, o plano de controle. Cada controlador possui seus módulos e lidam com os dados à sua maneira. O modelo apresentado na arquitetura representa uma extensão de um controlador com acréscimo dos módulos sugeridos para a implementação do serviço proposto. A seguir há um detalhamento desses módulos.

Coleta de elementos de rede e estatística: este módulo objetiva recolher informações da rede. Acessando módulos coletores de informações de um controlador, como *topology*, *discovery*, *route*, *device maneger*, etc. (relativos entre os controladores), coletam-se todos os dados necessários para uma consulta, tais como descoberta da topologia, dispositivos e seus respectivos dados estatísticos (por ex. perda de pacotes, número de pacotes recebidos e transmitidos). Os dados são atualizados sempre que houver alguma alteração na rede, seja inserção de novo switch, enlace ou solicitação de nova rota. Adicionalmente, dependendo do controlador, pode-se fazer consultas às tabelas de roteamento IP, ou qualquer outro módulo específico do controlador ao qual está sendo implementado o serviço.

Engenharia de tráfego: este módulo armazena as informações a respeito dos elementos de comutação, dos enlaces e suas estatísticas coletadas pelo módulo de coleta, construindo assim seu banco de dados de engenharia de tráfego (TED – *Traffic Engineering Database*), requisito necessário para configuração dos enlaces, mantendo uma visão topológica da rede. Os dados são armazenados em uma estrutura de grafos contendo todas as características que possam refletir a rede de forma real. Esse módulo é fundamental para a busca, pois é nele que são feitas as primeiras avaliações quanto à possibilidade de encontrar o caminho solicitado. Quando solicitado um caminho baseado em mais de um parâmetro, esse módulo é responsável por avaliar a possibilidade da criação desse caminho. Dessa forma, são verificados todos os nós e enlaces que atendam aos requisitos mínimos solicitados, por exemplo, número de pacotes perdidos, atraso, vazão, etc. Com esses dados criam-se as topologias customizadas (grafo customizado), a partir da topologia real. Caso o nó destino não pertença a essa topologia, uma mensagem de "caminho indisponível" é enviada à aplicação.

Grafo customizado: topologia que é armazenada de forma temporária em uma estrutura de grafos. O módulo Algoritmo de busca utiliza essa topologia para encontrar o caminho com custo mínimo. O objetivo é criar uma topologia contendo apenas os nós que atendam aos requisitos da aplicação.

Algoritmo de busca: é o componente responsável por computar o caminho, contendo um algoritmo de busca (que será apresentado na seção 3.2) denominado de GSearch, é capaz de encontrar o caminho baseando-se em um dos parâmetros apresentados que atenda aos requisitos solicitados pela aplicação.

Aprovisionamento de caminhos: módulo responsável por interagir com o administrador ou aplicação, na camada *northbound*, recebendo os requisitos necessários para a busca do caminho desejado ou consultas topológicas com informações de engenharia de tráfego. As consultas podem conter informações estatísticas de rede bem como a viabilidade de caminhos entre dois nós. Como função principal, esse componente faz o aprovisionamento de caminhos entre dois nós solicitados. A busca é efetuada a partir de requisitos passados pela aplicação. Como resultado, caso haja um caminho entre os nós, retorna ao solicitante (aplicação ou administrador da rede) um conjunto de nós referentes ao caminho encontrado, caso contrário, uma mensagem é enviada, informando a inexistência do caminho. De posse do caminho, pode-se criar um circuito fim a fim passando pelos nós informados.

3.2 Algoritmo de busca customizado (GSearch)

O módulo de Engenharia de tráfego é responsável pela geração dos grafos customizados, cujo algoritmo de busca é aplicado na dependência de quais requisitos serão utilizados pelo administrador ou aplicação. Portanto, a partir da topologia (grafo) real G <V, A> (sendo V referente aos vértices e A, às arestas), são gerados subgrafos, de tal forma que G1<V1, A1> pode ser um subgrafo, com base na largura de banda ou perda de pacotes. Outro subgrafo G2<V2, A2>, a partir de G1, é gerado quando se quer a combinação dos dois parâmetros. Logo, o dado de entrada para o algoritmo é: Gk<Vk, Ak>, sendo, Vk= 0, 1, 2,.., n; Ak= 0, 1, 2,.., m e k= 0, 1, 2. *K* corresponde ao número de subgrafos gerados e 0 corresponde ao grafo real. A medida que novos requisitos sejam necessários, novas combinações poderão ser geradas.

Posteriormente à geração da topologia customizada, é necessária a execução de um algoritmo para encontrar o menor caminho que esteja dentro do limite do parâmetro definido para a busca. Para responder a essa necessidade, é proposto o algoritmo GSearch, baseado no algoritmo de Dijkstra, para ser aplicado em qualquer dos K grafos. O caminho é sempre encontrado com base no menor valor do parâmetro entre origem e destino, levando-se em consideração peso das arestas, atraso, custo, etc. Esse caminho pode ser combinado com uma métrica (G1) ou duas (G2).

Para o cálculo do caminho, é necessário fazer uma somatória do peso d_i de cada enlace para verificar se a rota está dentro dos limites delimitados pela aplicação. Matematicamente é:

$$\pi(i) = \sum_{i=1}^{n} di \begin{cases} \pi(i) \geq L \ ; & \text{Solicitar nova rota} \\ \pi(i) < L \ ; & \text{Utilizar rota} \end{cases} \quad (1)$$

Sendo (1) a função matemática para calcular a viabilidade do caminho, n o número de enlaces pertencentes ao grafo, d_i o peso de cada enlace e L o valor de limite máximo permitido para a aplicação, e esse valor varia entre aplicações. Logo, se o valor da função for menor que o limitado pela aplicação, há uma caminho possível e esse caminho será retornado ao solicitante, caso esse valor seja maior ou igual ao limite da aplicação, o servidor irá informar que não há uma rota disponível para a configuração exigida.

O algoritmo de busca é apresentado na Tabela 1, utilizando o formalismo matemático, em que:

S: conjunto dos vértices já pesquisados;
R: conjunto dos vértices não pesquisados;
$\pi(i)$: custo do caminho entre o nó origem e o nó i;
$l_{i,j}$: custo da aresta (i, j);
$\Gamma(j)$: conjunto dos sucessores de j;
L: limite do atraso determinado para a aplicação ou pelo administrador.

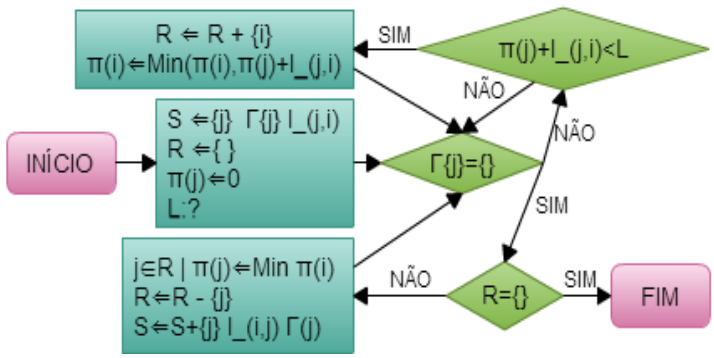
ALGORITMO 1 GSearch

Passo 0: Inicialização
Entrada: Grafo Customizado
$S \Leftarrow \{j\}\ \Gamma\{j\}\ l_{j,i}$
$R \Leftarrow \{\ \}$
$\pi(j) \Leftarrow 0$
$L \Leftarrow ?$
Passo 1: Determinar $\forall i \in \Gamma(j)$, faça:
Se $\pi(j) + l\_(j,i) < L$; faça
$R \Leftarrow R + \{i\}$
$\pi(i) \Leftarrow Min(\pi(i), \pi(j) + l_{i,j})$
Se $R = \{\ \}$; Fim
Senão; Ir ao passo 2;
Passo 2: $j \in R \mid \pi(j) \Leftarrow Min\ \pi(i)$; faça:
$R \Leftarrow R - j$
$S \Leftarrow S + \{j\}\ \Gamma(j)\ l_{j,i}$
Retornar ao Passo 1;

Na Figura 4, é apresentado um fluxograma que ilustra o funcionamento do algoritmo proposto. Primeiramente, mapeiam-se as variáveis, nas quais S recebe o nó inicial com seu conjunto de nós sucessores e seus respectivos enlaces; R inicializa com vazio, $\pi(i)$ com zero e L com o valor que identifica o limite de alcance permitido para a busca. No momento de inicialização da busca, é importante ressaltar que as variáveis os vértices devem inicializar com um valor máximo e que no decorrer da busca esse valor será alterado pela função $\pi(i) \Leftarrow Min(\pi(i), \pi(j) + l_{i,j})$.

Após a inicialização, tem-se o correspondente ao passo 1 do algoritmo, que determina todos os nós i alcançados a partir de j; caso a distância $\pi(j)$ somado ao valor do enlace para alcançar i seja menor que o valor limitado pela aplicação, deve-se inserir nó i em R e verificar se o valor atual de $\pi(i)$ é menor que a distância $\pi(j) + l_{i,j}$. Essa condição é importante para que apenas os nós que estão dentro do limite permitido pela aplicação sejam inseridos na lista de sucessores.

Figura 4: Fluxograma do algoritmo



Fonte: elaborado pelos autores

Após a verificação de todos os vértices alcançados a partir do inicial, verifica-se se R está vazio $(R = \{\})$; caso não esteja, inicia-se o processo referente ao passo 2 do algoritmo. Seleciona-se o vértice com menor custo $(\pi(i))$ pertencente ao conjunto R. Retira-se o vértice de R, inserindo-o em S com suas respectivas arestas. O conjunto S é fundamental para saber se o vértice destino foi encontrado ao final da execução da busca. Outro ponto implícito no formalismo, é que o nó selecionado no passo 2 pode guardar consigo seu antecessor, assim é possível encontrar o caminho com mais facilidade a partir do destino. Finalizando esse passo, retorna-se ao passo 1.

A condição de parada é R igual a vazio $(R = \{\})$, o que indica que não há mais elementos a serem percorridos. Deste modo, é possível afirmar que o caminho com o menor peso entre a origem e todos os demais nós do grafo, que estão dentro do limite estabelecidos pela aplicação, foram alcançados. Caso o nó destino esteja entre os alcançados, então o caminho desejado foi encontrado; caso contrário, informa-se que não há caminho disponível dentro das condições passadas.

Como afirmado anteriormente, o algoritmo é fundamentado no algoritmo de Dijkstra, cujo diferencial está no alcance da busca, o algoritmo fará uma pesquisa apenas nos vértices que possuem uma distância inferior ao limitado para a busca.

Considerando o pior dos casos, no qual o fator limite é maior que a distâncias entre a origem e todos os demais nós da rede, a complexidade do GSearch será igual à complexidade de Dijkstra. Segundo [10], Dijkstra possui duas complexidades distintas, dependendo unicamente de como é implementada a fila de prioridades, na qual, em uma implementação de sequência não ordenada, o tempo de execução corresponde a $O(n^2)$. Porém, caso utilize a implementação de *heap*, a complexidade é reduzida para $O((n+m)\,log\,n)$, em que n e m correspondem ao número de vértices e de arestas, respectivamente.

Além da complexidade supracitada, a arquitetura oferece dois diferenciais. O primeiro é que, dificilmente, trabalha com toda a topologia real, pois é gerada uma topologia customizada, que somente seria igual à original se a rede estivesse de acordo com todos os requisitos passados. O segundo diferencial diz respeito ao fato de que o limite máximo permitido por cada aplicação também pode reduzir o tempo de execução da busca (conforme Figura 8, mais adiante), considerando que todos os nós que estiverem a uma distância da origem superior ao limitado pela aplicação ficam fora do processamento, juntamente com seus enlaces, influenciando diretamente na complexidade do algoritmo. Logo, somente em casos extremos a complexidade será igual ao Dijkstra padrão.

4 Estudo de caso

O estudo de caso do presente artigo objetiva a apresentação de uma implementação da proposta, além de avaliar o impacto da utilização da arquitetura do serviço de cálculo de caminhos em SDN. Diante disto, foram realizadas simulações acerca da proposta.

As simulações foram realizadas para comparar o desempenho do algoritmo proposto, o Gsearch, com o algoritmo comumente empregado nos controladores atuais, o Dijkstra. Tal como na linguagem de programação, foi utilizado Java (JDK7). Dentre os controladores disponíveis foi utilizado o floodlight versão 0.91, que é uma implementação de controlador OpenFlow baseado em linguagem Java, com licença Apache, desenvolvido e mantido pela comunidade livre, incluindo engenheiros do Big Switch Networks. Adicionalmente, é um controlador bem estruturado e modularizado, facilitando a diferenciação entre os módulos propostos no trabalho.

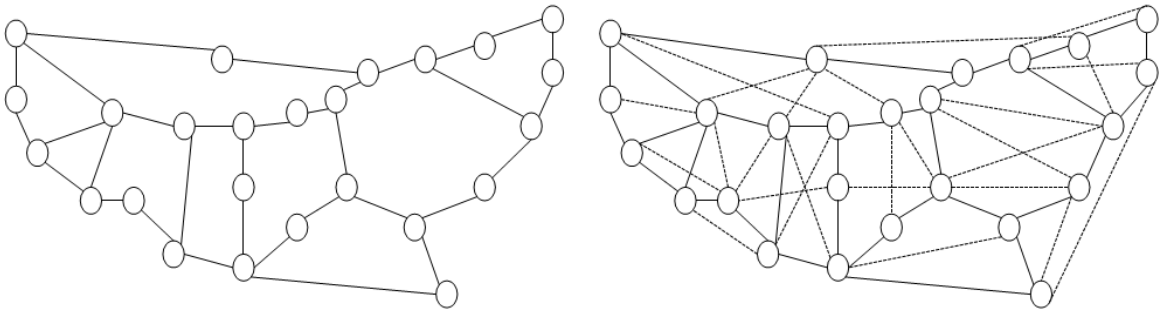
O ambiente utilizado para o desenvolvimento da aplicação e realização dos testes foi o Mininet.[5] Trata-se de um emulador de redes desenvolvido por pesquisadores da Universidade de Stanford, Estados Unidos, com o objetivo de apoiar pesquisas colaborativas, permitindo protótipos autônomos de SDN para que qualquer pessoa possa fazer o download, executar, avaliar, explorar e ajustar.

Para uma boa avaliação da proposta, quanto maior o número de vértices e arestas eleva-se a variabilidade de caminhos para uma solicitação. Por ser uma topologia complexa, com alto número de nós e enlaces, os testes foram efetuados em cenários baseados no backbone da internet2,[6] conforme ilustrado na Figura 5. O cenário 1 é a disposição original dos nós e enlaces disponíveis na topologia internet2. Paralelamente, o cenário 2 é baseado na topologia da internet2, que apresenta uma quantidade maior de enlaces, proporcionando mais variedade de

[5] Disponível em: <http://mininet.org/>.
[6] Disponível em: <http://www.internet2.edu/>

caminhos possíveis. O objetivo foi aliar dois casos de topologia: uma com pouca e outra com maior variedade de caminhos (Figura 5).

Figura 5: Cenários nos quais a proposta foi avaliada



Cenário 1 – internet2 **Cenário 2 - internet2 modificada**

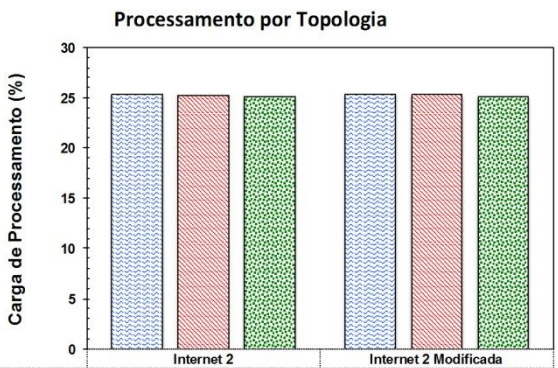
Fonte: elaborado pelos autores adaptado de internet2

A coleta dos dados foi efetuada em uma máquina com processador Intel(R) Core(TM) i5-3330@ 3,00GHz, Memória 8Gb DDR3, no qual tanto o controlador floodlight quanto a máquina virtual (virtual box) com o mininet estavam rodando. Com o objetivo de fazer solicitações entre todos os nós da topologia, foi desenvolvida uma aplicação que pudesse desempenhar tal função. Por conseguinte, solicitações foram geradas entre todos os nós de cada topologia; os dados a respeito do consumo de processador e memória *heap* foram coletados durante a execução.

Objetivando melhor avaliar o algoritmo proposto, ele foi submetido a duas avaliações distintas. A primeira foi chamada de Gsearch Limitado. Essa forma de avaliação representa a execução do algoritmo recebendo um valor que limita o alcance da busca, logo, apenas os nós que estiverem dentro do raio de alcance do limite serão visitados. O modo de avaliação Gsearch representa o algoritmo com um valor limite muito alto, no qual todos os nós do grafo estarão dentro do raio de alcance, sendo, pois, possível notar de forma expressiva a eficiência da proposta. Nessa primeira avaliação, levou-se em consideração o grafo completo, sem a geração dos subgrafos, objetivando avaliar a eficiência do algoritmo proposto sem considerar a totalidade da arquitetura.

Para que houvesse maior precisão dos dados, foram realizadas cinco execuções para cada algoritmo, considerando-se que cada execução corresponde a cerca de dez mil solicitações de caminhos entre todos os nós da topologia. Nos gráficos compreendidos nas figuras 6, 7 e 8, constam as médias dos dados coletados. Nas Figuras 6, 7 e 8, têm-se uma comparação entre o Dijkstra, Gsearch e Gsearch Limitado. Na Figura 6, observa-se a carga de processamento durante a realização dos experimentos; nota-se que, tanto para o cenário internet2 quanto para a internet2 modificada, a carga de processamento dos algoritmos são muito próximas, porém Gsearch e Gsearch Limitado consomem um processamento levemente inferior, se comparados ao Dijkstra.

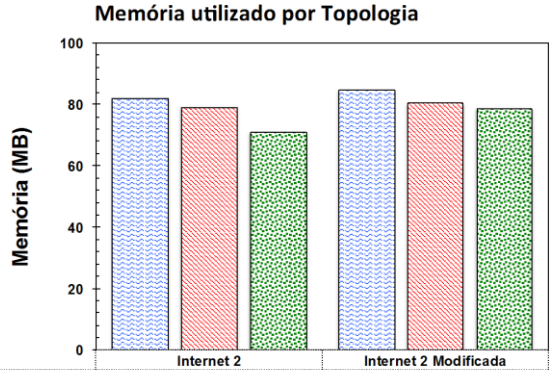
Figura 6: Carga de processamento



	Internet 2	Internet 2 Modificada
Dijkstra	25.31347	25.4002
Gsearch	25.27857	25.3001
Gsearch Limitado	25.09167	25.0750

Fonte: elaborado pelos autores

Figura 7: Consumo de memória *heap*



	Internet 2	Internet 2 Modificada
Dijkstra	82.1506	84.7701
Gsearch	79.0315	80.2882
Gsearch Limitado	70.9234	78.6147

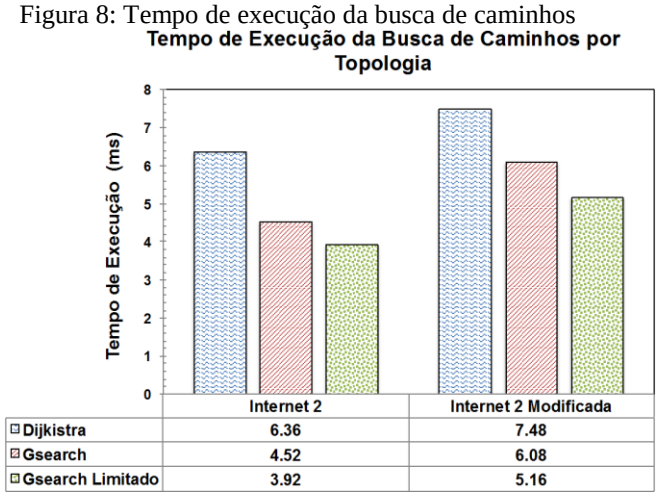
Fonte: elaborado pelos autores

Na Figura 7, é possível observar o consumo de memória *heap* durante as execuções. Destaca-se a eficiência do algoritmo, pois em ambas as versões do Gsearch houve menor consumo de memória. Vale ressaltar a observação da diferença nos resultados de consumo quando há utilização do fator limite. Isto pode ser atribuído ao fato de que nem todos os nós da topologia foram alcançados, reduzindo-se assim a utilização da memória *heap*.

No paradigma SDN, tem-se um modelo centralizado, no qual o controlador conterá todo o plano de controle da rede. Os resultados apresentados nas Figuras 6 e 7 demonstraram a redução de processamento e memória, o que é ideal para o controlador reduzir a sobrecarga de processamento e consumo de energia.

Na Figura 8, demonstra-se o tempo médio para o cálculo de caminhos entre os nós da topologia. Evidencia-se a eficiência da proposta, pois, como observado nos dados para internet2, que é uma topologia espaça e a mais comum entre as topologias reais, o tempo de processamento no Gsearch Limitado chega a ser aproximadamente 60% do tempo utilizado por Dijkstra. Mesmo em uma topologia com um maior número de arestas, o Gsearch apresenta-se com mais eficiência nas duas condições apresentadas.

Figura 8: Tempo de execução da busca de caminhos



Fonte: elaborado pelos autores

Dentre as contribuições do artigo, pode-se destacar: a criação de uma arquitetura aberta, que possibilitou o encontro do melhor caminho dentro de uma rede SDN, baseando-se em requisitos de QoS. Dessa maneira, há possibilidade de implementá-la em qualquer controlador; a combinação desses requisitos para encontrar o melhor caminho; e um algoritmo para o cálculo de caminhos de forma otimizada, que leva em consideração os requisitos de QoS da rede, cujo algoritmo apresenta uma discreta eficiência comparado a outro algoritmo consagrado pela literatura.

5. Conclusão e trabalhos futuros

As redes definidas por software ganham grande força nas redes de telecomunicações mundialmente. Contudo, a padronização de serviços permanece um grande desafio. Portanto, novas propostas de serviços nessa arquitetura mostram-se fundamentais para viabilizá-la.

Este trabalho atuou no desafio de oferecer um serviço de cálculo de caminho determinístico, ou não, cujas informações são repassadas pela aplicação. Usando-se como parâmetro determinístico a limitação do atraso. Definiu-se uma arquitetura a ser aplicada, a SDN, assim como o algoritmo a ser empregado no cálculo de caminhos.

Os resultados mostraram que o algoritmo proposto foi capaz de reduzir o tempo de busca de caminhos quando comparado ao algoritmo de Dijkstra. O Gsearch, avaliado no presente estudo, conseguiu reduzir o consumo de memória além do nível de processamento no controlador.

A partir dos resultados obtidos, testes adicionais empregando outras métricas de qualidade de serviço (por ex. variação do atraso e perda de pacote), assim como as características das redes Openflow (quantidade de regras nas tabelas ou números de fluxo em um comutador), tornam-se de relevante interesse. Pretende-se, pois,

realizar uma comparação com a aplicação *circuitpusher,* pertencente ao controlador floodlight, envolvendo todo o processo de criação de caminho da proposta descrita no presente estudo.

Referências

[1] CHOWDHURY, M.; BOUTABA, R. Network virtualization: state of the art and research challenges. *IEEE Communications Magazine*, v. 47, n. 7, p. 20-26, 2009.

[2] XIA, Wenfeng et al. A survey on software-defined networking. IEEE *Communications Survey & Tutorials*, v. 17, n. 1, p. 20-26, 2015.

[3] MCKEOWN, Nick et al. OpenFlow: enabling innovation in campus networks. *Sigcomm Comput. Commun*, v. 38, n. 2, p. 69-74, 2008.

[4] GREENBERG, Albert et al. A clean slate 4D approach to network control and management. *ACM Sigcomm Computer Communication Review*, v. 35, n. 5, p. 41-54, 2005.

[5] WICKBOLDT, Juliano Araújo et al. Software-defined networking: management requirements and challenges. *IEEE Communications Magazine*, v. 53, n. 1, p. 278-285, 2015.

[6] CASADO, Martin et al. Virtualizing the network forwarding plane. Sigcomm. 2010. Disponível em: <http://conferences.sigcomm.org/co-next/2010/Workshops/PRESTO/PRESTO_papers/06-Casado.pdf>. Acesso em: 17 jul. 2015.

[7] RAGHAVENDRA, Ramya; LOBO, Jorge; LEE, Kang-Won. Dynamic graph query primitives for SDN-based cloud network management. Sigcomm. 2012. Disponível em: <http://conferences.sigcomm.org/sigcomm/2012/paper/hotsdn/p97.pdf>. Acesso em: 17 jul. 2015.

[8] OPEN NETWORKING FOUNDATION. *Software-defined networking*: the new norm for networks. Open Networking Foundation. 2012. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf>. Acesso em: 14 ago. 2015.

[9] OPEN NETWORKING FOUNDATION. *OpenFlow switch specification Version 1.4*. Open Networking Foundation, 2013. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf>. Acesso em: 17 ago. 2015.

[10] GOODRICH, Michael T.; TAMASSIA, Roberto. *Projeto de algoritmos*: fundamentos, análise e exemplos da internet. Bookman Editora, 2009.