Universidade do Estado do Rio de Janeiro
Centro de Tecnologia e Ciências
Faculdade de Engenharia

Felipe Arruda Fernandes da Silva

Justiça de aplicações em redes definidas por software

Rio de Janeiro
2017

Felipe Arruda Fernandes da Silva

Justiça de aplicações em redes definidas por software

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre em Engenharia Eletrônica, ao Programa de Pós-Graduação em Engenharia Eletrônica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Redes de Telecomunicações.

Orientador: Prof. D.Sc. Marcelo Gonçalves Rubinstein
Orientador: Prof. D.Sc. Rodrigo de Souza Couto

Rio de Janeiro
2017

CATALOGAÇÃO NA FONTE

UERJ / REDE SIRIUS / BIBLIOTECA CTC/B

S586 Silva, Felipe Arruda Fernandes da.
Justiça de aplicações em redes definidas por software / Felipe Arruda Fernandes da Silva. – 2017.
48f.

Orientadores: Marcelo Gonçalves Rubinstein e Rodrigo de Souza Couto.
Dissertação (Mestrado) – Universidade do Estado do Rio de Janeiro, Faculdade de Engenharia.

1. Engenharia Eletrônica - Teses. 2. Arquitetura orientada a serviços (Computação) - Teses. 3. Interface de programas aplicativos (Software) - Teses. 4. Internet - Teses. I. Rubinstein, Marcelo Gonçalves. II. Couto, Rodrigo de Souza. III. Universidade do Estado do Rio de Janeiro. IV. Título.

CDU 005.1

Autorizo, apenas para fins acadêmicos e científicos, a reprodução total ou parcial desta tese, desde que citada a fonte.

Assinatura

Data

Felipe Arruda Fernandes da Silva

Justiça de aplicações em redes definidas por software

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre em Engenharia Eletrônica, ao Programa de Pós-Graduação em Engenharia Eletrônica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Redes de Telecomunicações.

Aprovada em 30 de Agosto de 2017.
Banca Examinadora:

Prof. D.Sc. Marcelo Gonçalves Rubinstein (Orientador)
Faculdade de Engenharia - UERJ

Prof. D.Sc. Rodrigo de Souza Couto (Orientador)
Faculdade de Engenharia - UERJ

Prof. D.Sc. Igor Monteiro Moraes
Instituto de Computação - UFF

Prof. Dr. Pedro Braconnot Velloso
Departmento de Engenharia Eletrônica e de Computação - UFRJ

Rio de Janeiro

2017

DEDICATÓRIA

Dedico este trabalho este trabalho a minha família, meus amigos e a meus orientadores, por terem me apoiado durante todo este caminho.

AGRADECIMENTOS

Agradeço aos meus pais por sempre me motivarem quando precisei e pela educação que recebi. Ao meu irmão por sempre me apoiar e estar sempre ao meu lado. Agradeço também à minha noiva pela compreensão e por me lembrar que tudo é possível. Aos meus orientadores pelos conhecimentos passados e por justificarem o que significa ser um orientador; obrigado pela paciência e por estarem sempre à disposição. Agradeço também aos membros da banca por dedicarem seu tempo a este trabalho e pela contribuição. Agradeço ao Programa de Pós-Graduação em Engenharia Eletrônica da UERJ pela oportunidade que me deram de fazer parte de um ambiente de pesquisa. Agradeço a todos os meus amigos que ficaram do meu lado e me ajudaram em momentos difíceis, em especial aos meus amigos de infância.

Fale pouco. Faça muito.
Benjamin Franklin

RESUMO

SILVA, F. A. F. S. *Justiça de aplicações em redes definidas por software*. 2017. 48 f. Dissertação (Mestrado em Engenharia Eletrônica) – Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2017.

Cada vez há mais aplicações que atendem a um grande número de usuários e que, além disso, são sensíveis a requisitos de Qualidade de Serviço (QoS), como a latência. Em uma aplicação *on-line* em tempo real, um requisito crítico de QoS é a latência entre os usuários e o servidor da aplicação. Como os usuários dessas aplicações podem estar geograficamente distribuídos, esses são afetados por níveis diferentes de latência com o servidor, fazendo com que cada usuário esteja sujeito a condições diferentes. Para aplicações *on-line* em tempo real, possuir atrasos diferentes pode levar a visões diferentes de um mundo digital, criando cenários injustos. Essa justiça é importante, por exemplo, em competições ou em algumas operações financeiras. Dessa forma, é necessário prover maior justiça entre os usuários dessas aplicações. As redes tradicionais possuem recursos limitados para lidar com essas exigências, entretanto vêm crescendo o emprego da tecnologia de redes definidas por software (*Software Defined Networking* - SDN), que possui um controle centralizado e facilita a implementação de aplicações altamente configuráveis. Com o controle centralizado, é possível analisar o desempenho de todos os usuários simultaneamente e aplicar as ações apropriadas. Por isso, a SDN torna mais fácil a obtenção de uma melhor justiça. O objetivo deste trabalho é propor uma aplicação de rede para SDN que permita atingir um maior grau de justiça entre os usuários de um dado serviço ou aplicação, garantindo que as latências entre cada um deles e o servidor da aplicação sejam as mais próximas possíveis. Para alcançar esse objetivo, o primeiro passo envolveu uma modelagem do problema, levando-se em consideração as latências atuais dos usuários e o caminho do fluxo de cada usuário. Utilizou-se um otimizador (CPLEX) para calcular a solução do problema e então, através de uma aplicação de rede proposta, foram empregados na rede os caminhos calculados e foram adicionadas latências nos *links* dos usuários, com o objetivo de obter um cenário mais justo possível. Para empregar na rede as ações sugeridas pela otimização, implementou-se uma aplicação SDN no controlador POX, que se comunica periodicamente com o CPLEX. Para avaliar o desempenho da aplicação utilizou-se o *Mininet* para a emulação de uma rede genérica e da rede da RNP. A aplicação de rede proposta foi comparada com soluções de comunicação mais tradicionais, como *hub* e *layer 2 learning*. Através dos experimentos, nota-se que a aplicação de rede proposta consegue obter um nível de justiça maior do que o das outras soluções.

Palavras-chave: SDN; QoS; Justiça;

ABSTRACT

SILVA, F. A. F. S. *Application fairness in software defined networking.* 2017. 48 f. Dissertação (Mestrado em Engenharia Eletrônica) – Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2017.

There are more and more applications serving a large number of users and that are also sensitive to Quality of Service (QoS) requirements. In a real-time online application, a critical QoS requirement is the latency between users and the application server. As users can be geographically distributed, they are affected by different levels of latency to the server, causing each user to be subject to different conditions. For on-line real-time applications, different delays may lead to different views of a digital world, creating unfair scenarios. This fairness is important, for example, in competitions or in some financial operations. So it is necessary to provide a better fairness between the users of these applications. Traditional networks have limited resources to deal with these requirements; however, Software-Defined Networks (SDNs), which have centralized control, facilitates the implementation of highly configurable applications. With centralized control, it is possible to simultaneously analyze the conditions of all users and apply appropriate actions. Hence SDN provides greater facility for obtaining better fairness. The objective of this work is to propose a network application for SDNs that allows achieving a greater degree of fairness among the users of a given service, ensuring that the latencies between each of them and the application server are as close as possible. To achieve this goal, the first step involves problem modeling, taking into account the users' current latencies and the path of the flow of each user. An optimizer (CPLEX) was used to calculate the solution of the problem and then, through our proposed network application, the calculated paths were implemented in the network and latencies were added in the users' links, in order to obtain the fairest scenario. In order to employ the actions in the network suggested by the optimization, an SDN application was implemented in the POX controller, which communicates periodically with CPLEX. To evaluate the performance of the application, *Mininet* was used for the emulation of a generic network and of the RNP network. The proposed network application was compared with more traditional communication solutions such as *hub* and *layer 2 learning*. Through the experiments, we conclude that the proposed network application achieves a level of fairness greater than that of the other solutions.

Keywords: SDN; QoS; Fairness;

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

DPID	*DataPath ID*
NOS	*Network Operating System*
POX	*Python NOS*
QoS	*Quality of Service*
RTT	*Round Trip Time*
SDN	*Software Defined Networking*

LISTA DE SÍMBOLOS

$\%$	Porcentagem
$\sum$	Somatório
$\in$	Pertence a
$\forall$	Para todos os valores de
$\geq$	Maior ou igual a
$=$	Igual a

SUMÁRIO

INTRODUÇÃO

Diversas aplicações na Internet, como *streaming* de vídeo, web, redes sociais e jogos *on-line*, estão com uma demanda alta e crescente (LI et al., 2015). Essas aplicações atendem a um grande número de usuários ao mesmo tempo e são sensíveis a variações na Qualidade de Serviço (*Quality of Service* - QoS) (GORLATCH; HUMERNBRUM, 2015). Um dos requisitos mais importantes relacionados à QoS é o atraso. No caso de jogos em tempo real, um jogador experiente é sensível a tempos de resposta muito baixos (p.ex., menores que 20 ms) (HUMERNBRUM; GLINKA; GORLATCH, 2014). No mercado de ações, é imprescindível que as ordens de compra e venda cheguem o mais rápido possível aos servidores, de forma a evitar prejuízos financeiros (MAXEMCHUK; SHUR et al., 2001).

Tentar alcançar um nível de QoS ideal para um serviço é um dos requisitos mais importantes a serem atendidos; por exemplo, não devem haver momentos em que a latência aumente drasticamente. A *Amazon* afirma que as suas vendas diminuem em 1 por cento a cada 100 ms de atraso no carregamento de uma página (KOHAVI; LONGBOTHAM, 2007). Os jogos *on-line* são um tipo de aplicação muito popular, apesar de serviços de QoS não estarem amplamente disponíveis na Internet (HENDERSON; BHATTI, 2003). Então pode se concluir que os jogadores geralmente se põem em condições de rede adversas. O efeito dos atrasos na rede é geralmente mais impactante para o jogador do que o *design* do jogo (como gráficos e som) e os efeitos exógenos (como limitações de hardware) e os jogadores parecem ser extremamente intolerantes às condições da rede (CHEN et al., 2006).

Segundo (BHAT et al., 2015), em pesquisas realizadas com jogadores visando descobrir quais os principais problemas (não excludentes, permitindo a marcação de mais de uma opção) percebidos por eles durante os jogos, 71,2% responderam "problemas com latência", 16,55% se referiram a "gráficos borrados", 30,4% informaram "queda de conexão" e o restante relatou não ter problema ou outros. Dessa forma é possível observar que o maior problema que os jogadores enfrentam em um jogo *on-line* está relacionado à QoS (BHAT et al., 2015).

Outro problema está relacionado à falta de informações sobre o comportamento dos usuários. A alocação dos servidores previamente realizada pelos provedores de serviço pode não ser a ideal e pode levar a uma qualidade inaceitável do serviço prestado. Isso ocorre, pois em alguns serviços, não se sabe de antemão quantos clientes utilizarão um determinado servidor, nem onde esses usuários estarão posicionados (LI et al., 2015). Consequentemente, a implantação dessas aplicações em uma infraestrutura em larga escala apresenta um desafio significativo. À medida que as distâncias geográficas entre os nós da rede aumentam, ocorrem maiores atrasos de propagação inevitáveis entre os participantes.

Isso pode tornar uma aplicação demasiadamente lenta para certos usuários, mesmo quando há recursos abundantes de processamento e recursos de rede disponíveis (BRUN; SAFAEI; BOUSTEAD, 2006).

Um requisito fundamental de todos os jogos *multiplayers* em tempo real é a consistência; ou seja, todos os jogadores devem compartilhar um mesmo "estado" do mundo digital; quando uma ação é realizada, como existe atraso entre o servidor e os jogadores, cada jogador tem uma visão diferente do estado. Certas aplicações tentam "prever" o movimento dos usuários caso haja falhas na rede. Logo, uma instabilidade da rede pode resultar em níveis completamente diferentes de percepção dependendo do tipo de ação do jogador (DEBROY et al., 2013). Por exemplo, um jogador que for utilizar uma arma de precisão requer uma alta QoS do sistema para que todos os jogadores possuam a mesma visão do mundo digital. Em certos cenários, as diferenças entre os tempos de resposta da aplicação aos diferentes usuários podem proporcionar a alguns usuários vantagens injustas (BRUN; SAFAEI; BOUSTEAD, 2006).

Atualmente, oferecer aos usuários uma garantia de QoS e também atender ao fornecedor do serviço é de importância primordial. No entanto, implementar tal QoS no sistema atual da Internet é um desafio. Isto é porque na Internet cada dispositivo de encaminhamento executa o seu próprio software com suas próprias regras, dependendo do protocolo utilizado. Além disso, o plano de controle está localizado em cada equipamento, criando um cenário em que cada dispositivo toma uma decisão "sozinho" e não há protocolo padrão disponível para configurar parâmetros de QoS nos dispositivos de encaminhamento (SHARMA et al., 2014).

Atualmente mudanças na Internet são demoradas, como a transição do IPv4 para o IPv6, iniciada há mais de uma década e ainda amplamente incompleta. Devido à inércia das redes IP atuais, um novo protocolo de roteamento pode levar cinco a dez anos para ser totalmente projetado, avaliado e implantado (KREUTZ et al., 2015). Uma tecnologia emergente, denominada redes definidas por software (SDN - *Software Defined Networking*), introduz um método diferente de implementar redes (KREUTZ et al., 2015). A SDN introduz o conceito de uma rede programável e permite que os sistemas finais alterem dinamicamente a topologia da rede, modificando assim a conectividade e o roteamento, e também possivelmente a QoS (HUANG; GRIFFIOEN, 2013). Essa tecnologia permite que seja possível criar uma rede customizada entre os usuários que estão requisitando um determinado serviço, por exemplo, ajustando a topologia da rede (através do uso de um servidor móvel) ou dando prioridade a certos fluxos (clientes que estão mais afastados recebem maior prioridade). Dessa forma, é possível oferecer uma melhor QoS sob demanda para os usuários; o ajuste da topologia da rede em tempo real não é trivial em outras tecnologias.

Motivação e Objetivos

A crescente popularidade dos esportes eletrônicos e a transmissão ao vivo de competições em redes de televisão a cabo aceleraram o crescimento de jogos online. Em 2001 quatrocentos jogadores de trinta e sete países diferentes competiram no *"World Cyber-Games"* por 300.000 dólares; já em 2006 a premiação foi de cinco milhões de dólares. Esse crescimento de premiação é um dos indicadores do quanto as empresas relacionadas a jogos evoluíram (JIN; CHEE, 2008). Em uma competição é de vital importância que todos os competidores tenham a mesma condição, a fim de que seja medida apenas a habilidade do competidor. Quando se trata de aplicações *on-line* é preciso levar em consideração a condição da rede dos usuários para que haja justiça. Para amenizar o efeito da rede, atualmente estes eventos são presencias. Porém com uma abordagem de maior justiça *on-line*, é possível viabilizar competições *on-line*.

O emprego de SDN possibilita criar aplicações de rede customizáveis e aplicar regras determinadas por uma entidade centralizada. O uso de centralização facilita a análise comparativa das condições de cada usuário e a aplicação das regras em vários pontos da rede simultaneamente.

Esta dissertação desenvolve uma aplicação de rede capaz de minimizar as diferenças entre as latências de vários usuários para um único servidor. Para isso utiliza-se um otimizador (CPLEX) responsável pela resolução de um problema de programação linear inteira mista (*Mixed Integer Linear Programming*), fornecendo os caminhos que o fluxo de cada usuário deve seguir e também um valor de atraso a ser adicionado no caminho de cada fluxo, com a finalidade de deixar as condições de uso da rede por cada usuário (em relação ao atraso) mais próximas possíveis umas das outras. A aplicação de rede é capaz de obter dados da topologia da rede e fornecer estas informações para o otimizador, com estes dados a aplicação de rede é responsável pela execução da saída do otimizador.

Para analisar a solução foram feitas medições de latências ao longo do tempo, variando-se o número de usuários em uma rede genérica com cinco nós e na rede acadêmica RNP. Estas redes foram emuladas através do *Mininet*, que possui suporte à SDN. Foram utilizados o método proposto (aplicação de rede) e dois outros métodos de uso mais comum, o *hub* e o *layer 2 learning* com *spanning tree*. Os resultados mostram que é possível atingir um nível de justiça maior, com o método proposto, e que para redes grandes, com muitos nós e *loops*, é possível ainda atingir latências menores do que as obtidas com os outros métodos testados.

Trabalhos relacionados

A fim de posicionar esta dissertação em relação aos trabalhos relacionados, os artigos são listados conforme suas contribuições. Esses estudos abordam aplicações interativas, SDN e justiça entre os usuários.

O estudo de (DEBROY et al., 2013) analisa gravações de vídeo de mais de cinco horas de um jogo, com dez voluntários, e identifica sessões nas quais a degradação da QoS causa inconsistência no estado do jogo, resultando em uma má experiência dos usuários. Também é mostrado o quanto a latência e a variação dela interferem no desempenho dos usuários.

Uma aplicação centralizada com controlador SDN é apresentada em (LI et al., 2015), com objetivo de obter controle centralizado de toda a rede e utilizando um modelo de otimização. Esta aplicação considera fornecer a melhor configuração possível, tanto para o provedor de serviço quanto para o usuário. Esta aplicação de rede é um balanceador de carga em que os fluxos são redirecionados para os servidores que estão sendo menos utilizados naquele momento. Além disso, de forma semelhante a esta dissertação, o trabalho aplica a solução de um problema de otimização em uma rede SDN.

Em (GORLATCH; HUMERNBRUM, 2015) são utilizados os recursos de gerenciamento dinâmico da SDN para expressar, monitorar e controlar as demandas de QoS das aplicações. O trabalho desenvolveu uma API *northbound* para atender aos requisitos de QoS de uma aplicação interativa. Além disso, demonstrou-se como uma métrica da aplicação, tempo de resposta, pode ser traduzida automaticamente em métricas de rede. Por fim, o trabalho apresentou resultados experimentais relacionados aos requisitos de QoS a nível de aplicação, utilizando um exemplo de jogo online implementado no *OpenFlow*.

O trabalho de (HUANG; GRIFFIOEN, 2013) propõe uma aplicação denominada *HyperNet* que interage com um jogo, sendo capaz de criar e implantar dinamicamente uma SDN que se adaptada às necessidades do jogo e aos seus participantes. Os autores também mostram como a aplicação ajuda na customização da rede quando comparada com uma rede pública e que o RTT de um usuário diminui consideravelmente, de 212 para 22 ms.

O artigo (BRUN; SAFAEI; BOUSTEAD, 2006) apresenta uma visão geral de vários fatores que podem afetar a qualidade da experiência do jogo em termos de jogabilidade e justiça. O trabalho mostra como a distribuição geográfica e as inconsistências na visão de um mesmo mundo podem impactar na justiça. Além disso, é demonstrado que a seleção cuidadosa e a organização geográfica de servidores de jogos podem ser de grande valor na melhoria da jogabilidade e da justiça. Os resultados mostram como a mudança no nível de justiça pode afetar na jogabilidade.

Vários artigos abordam aplicações diferentes de SDN para casos específicos e va-

riados. Isso é explicado pela grande facilidade de implementação e, consequentemente, de inovação das redes SDN. Porém, o trabalho apresentado possui uma abordagem única relacionada à justiça em termos de latência entre vários usuários, na qual a justiça é provida a partir da execução de uma entidade central, o que permite uma análise mais fácil do cenário completo ao mesmo tempo. Este modelo permite que competições possam ser feitas *on-line* com justiça.

Organização do Texto

Esta dissertação está estruturada da seguinte forma. O Capítulo 1 apresenta os conceitos básicos de SDN, como a sua arquitetura e seu funcionamento. No Capítulo 2, é apresentada a formulação do problema de otimização, descrevendo a função objetivo e as restrições adotadas. A implementação do problema é discutida no Capítulo 3, no qual é discutido como a aplicação de rede fornece os parâmetros para o CPLEX, e as respostas da mesma são aplicadas na rede física. O Capítulo 4 apresenta os experimentos realizados e os resultados dos testes. Por fim, esta dissertação é concluída no Capítulo 5.

1 REDES DEFINIDAS POR SOFTWARE

Há crescimento rápido na demanda por equipamentos SDN, fazendo com que fabricantes de equipamentos de rede adicionem suporte a SDN (HUANG; GRIFFIOEN; CALVERT, 2014). Com essa integração é possível criar equipamentos que continuam podendo atuar na rede tradicional, o que permite uma evolução paulatina da rede.

Os nós da rede tradicional geralmente são organizados em três planos: dados, controle e gerenciamento. O plano de dados é responsável por encaminhar todas as mensagens que são geradas pelos usuários. Para transportar essas mensagens, a rede precisa realizar ações, como encontrar o caminho mais curto usando os protocolos de roteamento da terceira camada, como o *Open Shortest Path First* (OSPF) ou protocolos na segunda camada, como o *Spanning Tree*, tarefa a cargo do plano de controle. As mensagens usadas para este propósito são chamadas de mensagens de controle e são essenciais para a operação da rede. Além disso, o gerente de rede pode querer acompanhar as estatísticas de tráfego e o estado dos vários equipamentos de rede. Isso é feito através do gerenciamento de rede. O gerenciamento, embora importante, é opcional e muitas vezes não é usado em redes pequenas, como as redes domésticas (JAIN; PAUL, 2013).

Uma das principais inovações do SDN corresponde ao desacoplamento do plano de controle do plano de dados, retirando o plano de controle dos dispositivos tradicionais de rede (*switches* e roteadores). O plano de dados encaminha os pacotes usando tabelas de encaminhamento geradas a partir de informações fornecidas pelo plano de controle. A lógica de controle é separada e implementada em um controlador. Os *switches*, através das informações repassadas pelo controlador, implementam a lógica do plano de dados (encaminhamento) que é bastante simplificada. Isto reduz significativamente a complexidade e o custo dos *switches*. Estas tabelas de repasse são chamadas de tabelas de fluxos (*flow tables*); nelas estão indicadas as regras para cada fluxo determinadas pelo controlador. Essas regras representam ações a serem tomadas como: descarte do pacote (*dropping*), repasse, modificação, etc. De acordo com o conjunto de regras na tabela de fluxos, o dispositivo pode atuar com múltiplas funções, pois podem haver regras com que o *switch* atua tanto como *firewall* e balanceador de cargas ao mesmo tempo, por exemplo.

1.1 A arquitetura SDN

A SDN pode ser definida como uma arquitetura de rede com quatro pilares (KREUTZ et al., 2015):

- os planos de controle e dados são desacoplados;

- ao contrário das redes tradicionais, na SDN as decisões de encaminhamento são baseadas em fluxo, e não apenas no endereço de destino. Um fluxo corresponde a um conjunto de valores de campos do cabeçalho de um pacote usado na verificação de correspondências com uma tabela (filtragem) e para o qual é executado um conjunto de ações (instruções). Para SDN, um fluxo é uma sequência de pacotes, com a mesma identificação, entre uma fonte e um destino. A abstração do fluxo permite unificar o comportamento de diferentes tipos de dispositivos de rede, incluindo roteadores, *switches*, *firewalls* e *middleboxes* (JAMJOOM; WILLIAMS; SHARMA, 2014). A programação de fluxo cria alta flexibilidade, limitada apenas pelas capacidades das tabelas de fluxos implementadas;

- a lógica de decisões é movida para uma entidade externa, o intitulado controlador SDN ou sistema operacional de rede (*Network Operating System - NOS*). O NOS é uma plataforma de software que fornece recursos essenciais e abstrações para facilitar a programação dos dispositivos de encaminhamento. Sua finalidade se assemelha a de um sistema operacional tradicional;

- a rede é programável através das aplicações de rede executadas sobre o NOS que interage com os dispositivos do plano de dados. Essa é uma característica fundamental da SDN, considerada como a principal proposta da tecnologia SDN.

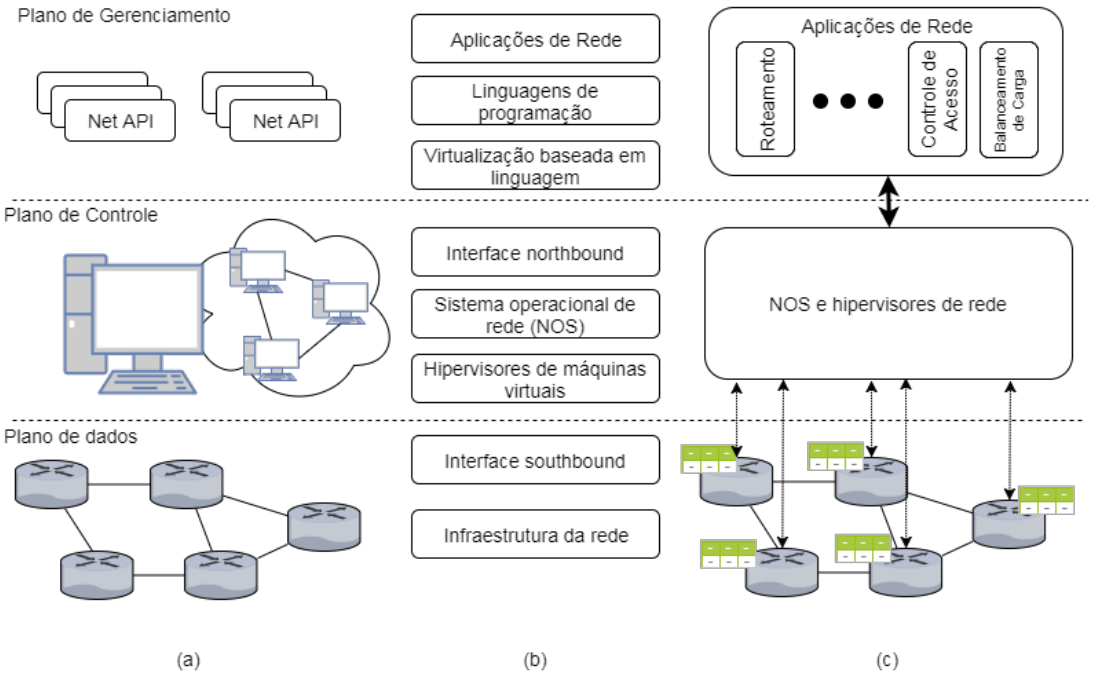
A Figura 1 representa a SDN em três diferentes visões: os planos (a) como discutidos anteriormente, as camadas (b) e a arquitetura (c).

As camadas do plano de dados se referem à infraestrutura da rede e à interface *southbound*, estando diretamente relacionadas aos dispositivos físicos. A infraestrutura da rede corresponde aos dispositivos de repasse e suas ligações físicas, como os equipamentos de uma rede tradicional (*switches*, roteadores, etc.); porém a diferença principal é que na rede SDN esses equipamentos não possuem "inteligência" e as decisões são tomadas no plano de controle. Exemplos de *switches* de implementações em SDN incluem *Switch Light*, *ofsoftswitch13*, *Open vSwitch*, *OpenFlow Reference*, *Pica8*, *Pantou* e *XorPlus*. A interface *southbound* é um software responsável pela conexão entre os dispositivos de repasse e o plano de controle, sendo crucial para a separação dos planos. Atualmente a interface *southbound* mais implementada é o *OpenFlow*. O protocolo *OpenFlow* fornece três tipos de informação para *NOS*:

- as mensagens baseadas em eventos são enviadas para o controlador pelos dispositivos de repasse quando há uma mudança de link ou porta;

- as estatísticas de fluxo são geradas pelos dispositivos de repasse e coletadas pelo controlador;

Figura 1 - A SDN representada em (a) planos, (b) camadas e (c) arquitetura.



Fonte: Adaptado de KREUTZ et al, 2015.

- as mensagens *packet in* são enviadas para o controlador quando o dispositivo de repasse não sabe o que fazer com um novo fluxo de entrada ou porque há uma ação de "enviar para o controlador" explícita na entrada correspondente da tabela de fluxo.

Cada camada na rede SDN, representada na Figura 1b possuem funções importantes para o funcionamento em geral.

Os hipervisores permitem que máquinas distintas compartilhem os mesmos recursos de hardware. Da mesma forma, o hipervisor de rede permite que diversas aplicações da rede compartilhem a infraestrutura física. Os sistemas operacionais tradicionais fornecem abstrações para acessar dispositivos de nível inferior e gerenciam o acesso concorrente aos recursos físicos. Analogamente para os sistemas computacionais, o hipervisor é responsável pelas abstrações dos dispositivos de repasse para as aplicações de rede e também gerenciam o acesso concorrente aos recursos físicos. Essas funcionalidades e recursos foram essenciais para o desenvolvimento de sistemas e aplicativos, em contraste os dispositivos de rede tem se mantido principalmente com instruções de baixo nível. Exemplos de hipervisores são: *FlowVisor, FlowN, NVP* e *RadioVisor*.

O NOS possui a função de abstrair características específicas dos dispositivos de rede, fornecendo funcionalidades comuns de forma transparente; para que, por exemplo, desenvolvedores de protocolos de roteamento não precisem lidar com algoritmos dis-

tribuídos complicados ao resolverem problemas de rede.

A interface *northbound* é crucial para promover a interoperabilidade das aplicações entre as diferentes plataformas de controle. Ou seja, essa interface garante a comunicação entre controladores de diversos tipos com diferentes linguagens de programação utilizada. Não há nenhuma interface *northbound* bem definida, atualmente cada desenvolvedor do controlador cria sua própria.

Duas características essenciais da virtualização baseada em linguagem são a capacidade de expressar modularidade e de permitir diferentes níveis de abstrações garantindo as propriedades desejadas, tais como proteção.

As técnicas de virtualização podem permitir visões diferentes de uma única infraestrutura física; por exemplo, é possível que um "*switch* grande" virtual poderia representar uma combinação de vários dispositivos de encaminhamento subjacentes.

As linguagens de programação têm evoluído com o tempo, migrando do baixo nível, como o Assembly para o alto nível, como o Phyton. As abstrações fornecidas por linguagens de programação de alto nível podem ajudar significativamente a resolver muitos dos desafios, como criar abstrações de alto nível para simplificar a tarefa de programação de dispositivos de encaminhamento e disponibilizar ambientes mais produtivos e focados para programadores de softwares de rede (HINRICHS et al., 2009).

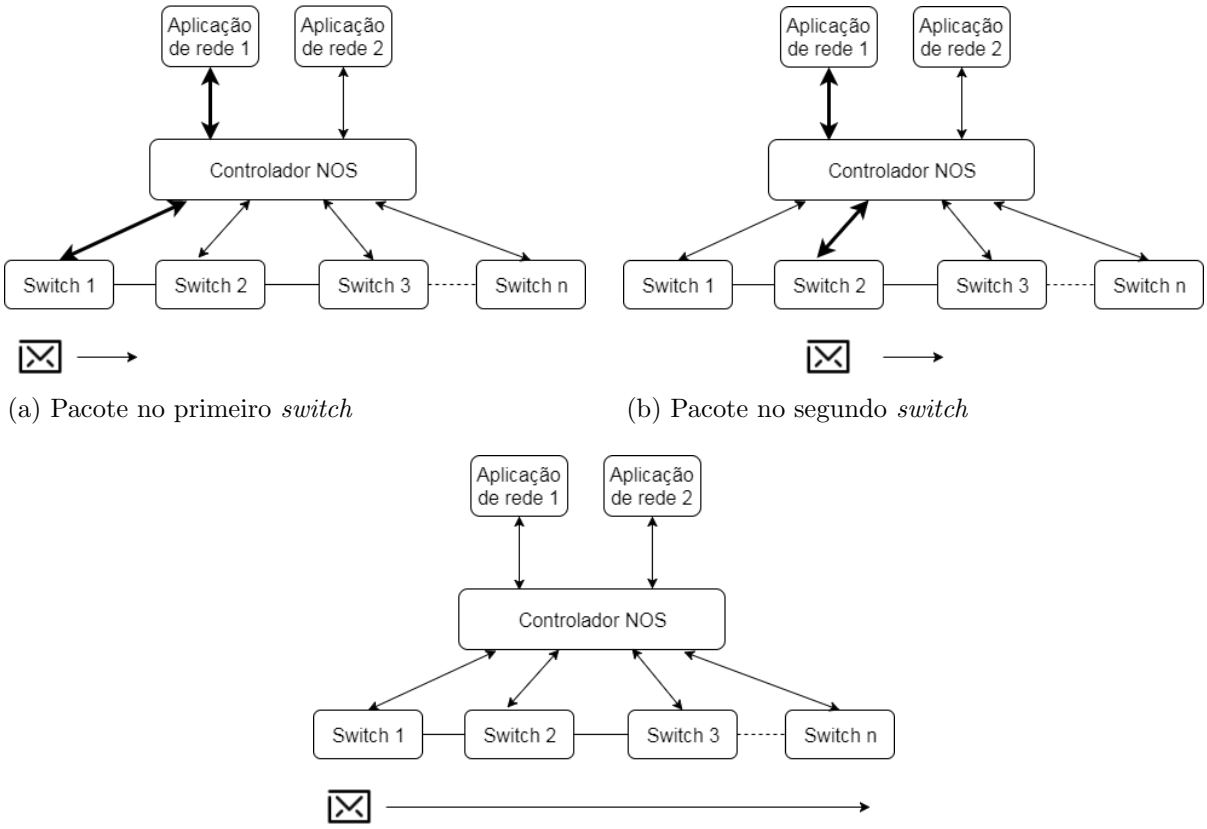
As aplicações de rede (Net APIs – *Net Application Programming Interfaces*) podem ser consideradas os "cérebros da rede". Elas implementam a lógica de controle que será traduzida em comandos a serem instalados no plano de dados, ditando o comportamento dos dispositivos de encaminhamento. As aplicações de redes existentes executam funcionalidades tradicionais, tais como roteamento, balanceamento de carga e implementação de políticas de segurança, mas também podem explorar novas abordagens, como a redução do consumo de energia.

1.2 O funcionamento de uma rede SDN

Embora a arquitetura SDN é composta de várias camadas, seu funcionamento é bem simples. A visão de arquitetura da Figura 1 representa as entidades principais do funcionamento de uma rede SDN.

Em seguida ao modelo em camadas de SDN, a Figura 2 mostra o funcionamento prático de uma rede desse tipo. Em uma rede não configurada, quando um pacote chega ao primeiro *switch*, fato representado na Figura 2a, o *switch* 1 não possui regra instalada em sua tabela de fluxos, ou seja, não há informação sobre o que fazer com este pacote. Então é gerada uma mensagem denominada *packet in* para o NOS. O NOS identifica o *packet in* e então o repassa para a aplicação de rede correspondente (*Net API* 1) que, por sua vez, informa a ação que deve ser tomada para o NOS, fato também mostrado na

Figura 2 - Exemplo de funcionamento de uma SDN.



(a) Pacote no primeiro *switch*

(b) Pacote no segundo *switch*

(c) Rede configurada

Figura 2a. Essa ação é então instalada na tabela de fluxos do *switch* 1.

A Figura 2b indica que o pacote foi repassado do *switch* 1 para o *switch* 2, porém quando o pacote chega ao *switch* 2, não há regra na tabela de fluxos para o este *switch*, então o mesmo processo é realizado. Este mesmo processo ocorre também para todos os dispositivos de repasse, até que o pacote chegue no seu destino.

O próximo pacote do fluxo que chegar ao primeiro dispositivo encontrará a regra já instalada na tabela de fluxos, então o *switch* 1 repassa o pacote sem precisar perguntar para a aplicação qual a ação a ser tomada. Acontecerá o mesmo para todos os *switches* em que o fluxo passará, pois o fluxo seguirá o mesmo caminho do primeiro pacote do mesmo. Logo, todo o resto do fluxo irá ser processado rapidamente, fato representado pela Figura 2c.

As ações tomadas no exemplo anterior foram somente de repasse, porém há outros tipos de ação. Em uma aplicação de *firewall*, por exemplo, ações de descarte de pacote podem ser também instaladas na tabela de fluxo.

A configuração da tabela de fluxos de um *switch* permanecerá pela duração de um tempo de expiração (*timeout*). Este tempo é definido pela aplicação de rede. A configuração pode terminar também por uma ação do controlador de limpeza das tabelas

de fluxo, porém quando esta ação é tomada, todas as regras são apagadas, sendo necessária a reconfiguração de todos os fluxos novamente.

2 MODELAGEM DO PROBLEMA

O objetivo deste trabalho é propor um modelo em que se existe uma condição de rede mais justa em uma aplicação com um servidor e vários usuários. Esta justiça se refere à latência, ou seja, ao atraso entre o usuário e o servidor. Como apresentado anteriormente é preciso consistência nas informações que chegam aos usuários e o fato de um ou mais usuários possuírem menores latências em relação aos outros usuários causa um cenário injusto. Em uma situação de competição, tentar fazer com que todos os usuários estejam sujeitos às mesmas condições é essencial. Por esta razão, este trabalho tem como objetivo propor uma condição de rede em que todos os usuários tenham a mesma latência.

No cenário utilizado há uma rede SDN aonde todos os dispositivos de repasse se comunicam com um controlador SDN, e os usuários se conectam nesta rede através das redes tradicionais, e o servidor se encontra dentro da rede SDN; como mostrado na Figura 3. Nesta rede há vários fatores que influenciam a latência total do usuário, porém os principais são o caminho que o fluxo segue e o atraso de propagação do usuário fora da rede SDN. Logo, em uma rede é raro os usuários possuírem as mesmas latências. A aplicação proposta calcula os caminhos dos usuários e adiciona atrasos nos usuários para que as latências dos usuários sejam matematicamente iguais. As soluções deste problema serão obtidas através de programação linear.

Para uma melhor análise dos usuários, que facilita criar um cenário mais justo, simultaneamente foi escolhida a plataforma SDN, por possuir o sistema de controle centralizado. Como a decisão sobre o fluxo em uma rede SDN é determinada por uma aplicação de rede, o objetivo então é criar uma aplicação de rede que execute o problema e seja capaz de aplicá-lo. Logo, esta aplicação de rede será capaz de ler os parâmetros da rede para fornecê-los ao otimizador, posteriormente interpretar as soluções e aplicá-las, ou seja, os fluxos de cada usuário devem seguir os caminhos calculados, e inserir os atrasos

Figura 3 - Cenário Utilizado.

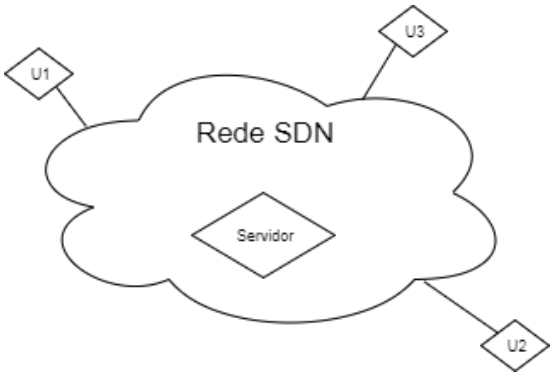


Tabela 1 - Notações utilizadas no problema.

Notação	**Descrição**	**Tipo**
S_k	Nó de entrada do usuário k na rede	Parâmetro
$Sink$	Nó em que o servidor está localizado	Parâmetro
C_{ij}	Custo (latência) entre os nós i e j	Parâmetro
l_k	Latência do usuário k até o nó de entrada na rede indicado por S_k	Parâmetro
D	Dispositivos de repasse na rede SDN	Conjunto
K	Usuários	Conjunto
E_{ijk}	Utilização do *link* ij pelo usuário k	Variável Booleana
$\triangle_k$	Latência introduzida para o usuário k	Variável
r	Latência total de todos os usuários	Variável

equivalentes de cada usuário.

A modelagem do problema abordado se assemelha à utilizada no problema de cálculo de fluxo em uma rede tradicional (GOLDBARG; LUNA, 2005). A grande diferença entre esse problema e a abordagem tradicional é a necessidade de análise de vários usuários simultaneamente, para que seja possível a comparação entre eles e o estabelecimento da justiça. No problema também são considerados os atrasos dos usuários fora da rede SDN e a latência que será inserida.

2.1 **Função objetivo**

Para uma maior justiça, ou seja, todos os usuários possuem o mesmo valor de latência; logo é preciso criar uma função que englobe a latência de todos os usuários. Então a variável r é a latência total (soma de todas as latências entre um usuário e o servidor) ideal de um usuário, igual para todos os usuários. A igualdade é possível pelo fato de a variável $\triangle_k$ ser utilizada na formulação, de forma a tornar a latência total igual para todos os usuários. Essa variável pode assumir qualquer valor inteiro; logo, para cada usuário, $\triangle_k$ é utilizado para que todos os usuários possuam um mesmo valor de latência total. Porém, mesmo que seja adicionado atraso, ainda é importante minimizar o pior caso. A função objetivo do problema é representada pela Equação 1, na qual minimizando-se a variável r, garante-se a justiça com a menor latência.

A Tabela 1 mostra as variáveis e as constantes utilizadas no problema.

A variável r é igual a soma das latências do caminho, do usuário até a rede SDN (l_k) e da latência introduzida ($\triangle_k$). A variável $\mathcal{E}_{ijk}$ indica se o usuário k está usando um dado *link* entre o nó i e o nó j. Note que há uma diferença entre $\mathcal{E}_{131}$ e $\mathcal{E}_{311}$, já que i indica a origem do fluxo no *link*. O somatório na Equação 2 representa então o custo total do caminho de um usuário, pois cada link utilizado representado por $\mathcal{E}_{ijk}$ é multiplicado pela latência equivalente do link e isso corresponde à latência total do caminho. A seguir são apresentadas as equações relacionadas ao problema e, em seguida, breves discussões

sobre cada uma.

$$\text{minimizar } r \tag{1}$$

$$\text{sujeito a} \quad r = l_k + \triangle_k + \sum_{i,j \in D} \mathcal{E}_{ijk} \cdot C_{ij}, \forall k \in \mathcal{K} \tag{2}$$

$$\sum_{j \in D} \mathcal{E}_{ijk} - \sum_{j \in D} \mathcal{E}_{jik} = 0, \forall k \in \mathcal{K}, \forall i \neq \{Sink, S_k\} \tag{3}$$

$$\sum_{j \in D} \mathcal{E}_{ijk} - \sum_{j \in D} \mathcal{E}_{jik} = 1, \forall k, i = S_k \tag{4}$$

$$\sum_{j \in D} \mathcal{E}_{ijk} - \sum_{j \in D} \mathcal{E}_{jik} = -1, \forall k, i = Sink \tag{5}$$

$$\sum_{i \in D} \mathcal{E}_{ijk} = 1, \forall k \in \mathcal{K}, \ \forall j \in \mathcal{D}, \tag{6}$$

$$\triangle_k \geq 0 \tag{7}$$

$$\mathcal{E}_{ijk} \in \{0, 1\} \tag{8}$$

As Equações 3, 4 e 5 dizem respeito ao caminho de cada fluxo de cada usuário; ou seja, na análise o caminho do fluxo se refere a apenas um usuário. A relação que os fluxos possuem entre eles é o custo do caminho total que contribui na variável r.

A Equação 3 caracteriza que em cada nó o somatório dos fluxos de entrada devem ser iguais ao somatório dos fluxos de saída, porém esta equação não é válida para os nós em que a origem é o usuário ou o destino é o *Sink*; logo, a equação é válida para os nós intermediários do fluxo. Quando um *link* será usado e $\mathcal{E}_{ijk} = 1$ para que a Equação 3 se torne verdadeira algum link que com ponto de origem j tem de ser usado, garantindo assim o encaminhamento; por exemplo, se o *link* $\mathcal{E}_{121} = 1$ algum *link* com origem no nó

"2" terá de ser igual a "1", dando assim continuidade ao fluxo.

As Equações 4 e 5 são referentes ao início e fim de fluxo. A Equação 4 estabelece que só pode haver um link que esteja conectado à origem do fluxo, como $\mathcal{E}_{ijk}$ só pode assumir valores binários (Equação 8), isso garante que a origem do fluxo será o nó de entrada do usuário. O análogo é válido para a Equação 5 para o *Sink*, porém o fluxo deve chegar neste nó, ou seja, onde se encontra o servidor. Logo, as duas equações garantem que o caminho comece sempre no usuário referente e termine no servidor.

A Equação 6 define que apenas um fluxo, por usuário, possa chegar em cada nó, pois se um link for escolhido para ser utilizado, nenhum outro link com a mesma origem pode ser escolhido. Como o problema não é o de "menor custo", o otimizador pode escolher um caminho convencionalmente "não eficiente" e uma das possibilidades para este caso é possuir fluxo saindo e depois chegando no mesmo nó, ou vice-versa. Por exemplo, seria possível ter um caminho S_k-1-2-$Sink$-3-$Sink$ ou S_k-1-2-3-1-$Sink$, o que não é possível em SDN, pois não se pode aplicar duas regras para o mesmo fluxo no *switch*.

Por último, a Equação 7 determina que o valor da latência introduzida para o usuário k deve ser maior ou igual a 0.

3 IMPLEMENTAÇÃO

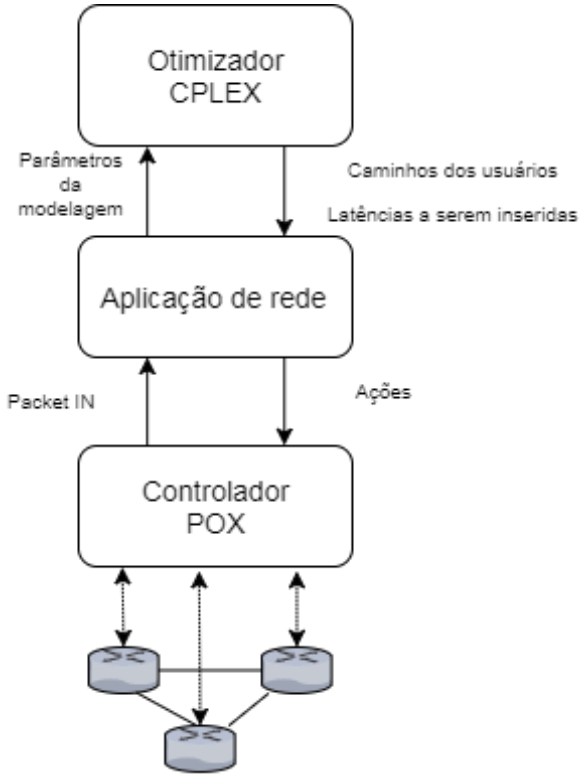
O problema de otimização formulado visa oferecer maior justiça entre usuários dentro de uma rede definida por software, como visto no Capítulo 2. A abordagem por SDN foi escolhida principalmente por possuir um plano de controle centralizado, possibilitando tomar decisões para todos os usuários e nós simultaneamente. Como a solução do problema é realizada por apenas uma entidade, o controlador, a execução da solução em toda a rede é facilitada.

A resolução do problema de programação linear é realizada através do CPLEX; um software de otimização de alto desempenho (ACHTERBERG; BERTHOLD, 2007). Porém, é necessária a implementação de uma aplicação de rede no controlador SDN que seja capaz de interagir com o CPLEX, providenciando os parâmetros do problema a ser resolvido e interpretando a saída do CPLEX.

Como o problema foi implementado em *Python*, escolheu-se também desenvolver a aplicação de rede na mesma linguagem de programação, com a finalidade de melhor interação entre as duas entidades. Com isso, para implementar as ações determinadas pelo otimizador, por exemplo, aumentar a latência para um determinado usuário, foi utilizado o controlador POX (*Python NOS*). O POX foi escolhido por ser implementado em Python e por ter seu código aberto no GitHub (GITHUB, 2017). Seu desenvolvimento possui um foco maior na melhor integração na interface do que manter uma aplicação estável (AZODOLMOLKY, 2013). O POX é então responsável pela execução das ações que serão requisitadas pela aplicação de rede. A comunicação entre os componentes da solução proposta está representada na Figura 4.

A aplicação de rede desenvolvida tem a função de ser um módulo intermediário entre o POX, que irá executar as ações, e o otimizador, no qual serão calculados os caminhos dos fluxos dos usuários para o servidor. Porém, antes da execução do otimizador, a aplicação de rede precisa realizar uma descoberta da rede e identificar os endereços de cada componente, e então fazer um reconhecimento sobre os parâmetros da rede. Após a etapa de otimização, é realizada a configuração dos caminhos dos fluxos. Esse processo está representado na Figura 5: no estado de descoberta a aplicação de rede salva as informações que são constantes na rede, como endereços e portas físicas utilizadas. Depois de coletados os dados dos componentes da rede é feito o reconhecimento dos custos de cada *link* e os outros parâmetros necessários, apresentados no Capítulo 2 (estado de reconhecimento). Então é feita a otimização do problema. No estado de configuração são salvos os caminhos dos fluxos de cada usuário, para serem configurados nas tabelas de fluxo e também aplicado a latência inserida (caso haja). Depois da configuração, realiza-se periodicamente o reconhecimento da rede, pois podem haver alterações nas latências dos *links*, devido a tráfegos, por exemplo. Sendo então necessária uma reconfiguração, então

Figura 4 - Trocas de informações entre os componentes da solução proposta.



volta-se o estado da aplicação de rede para a etapa de reconhecimento.

3.1 A arquitetura da aplicação de rede

A Figura 6 apresenta o diagrama de blocos da aplicação de rede e como ela interage com as entidades externas, que são: os usuários, os *switches* e o controlador POX. A aplicação de rede interage diretamente com os usuários e os *switches* através do "socket de mensagem" e indiretamente com os *switches* através do POX, devido à abstração que

Figura 5 - Estados da aplicação de rede.

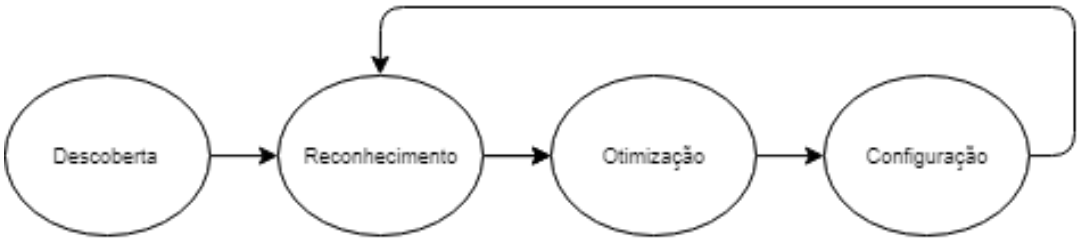
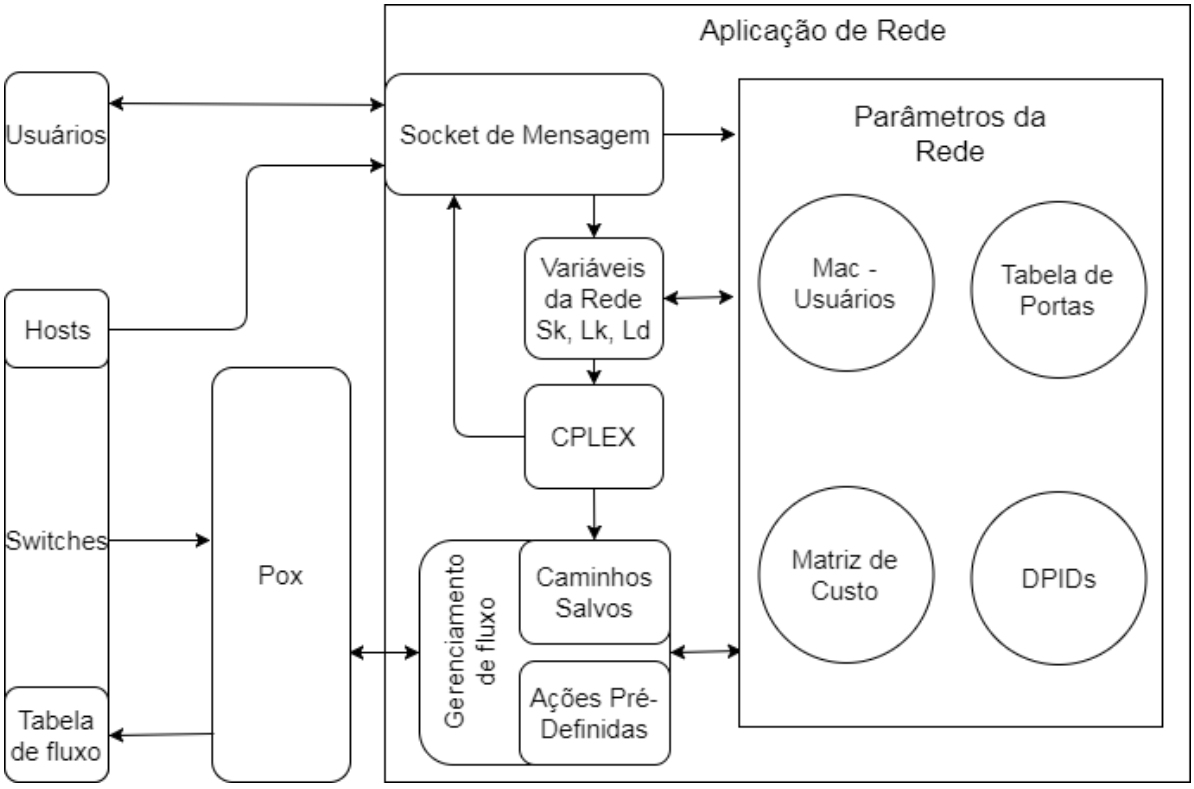


Figura 6 - Diagrama de blocos da aplicação de rede.



o POX realiza da rede.

O socket de mensagem recebe as mensagens dos usuários e dos *switches* e também envia para os usuários mensagens correspondendo a latências a serem adicionadas no enlace entre o usuário e o primeiro *switch* pelo qual passa o fluxo do usuário. Através desse *socket* também são recebidas mensagens relativas às medições das latências entre os nós da rede que são armazenadas nos parâmetros de rede, na "matriz de custo".

Os "parâmetros da rede" correspondem a tabelas e a uma matriz que contém os dados da rede que a aplicação irá utilizar. A tabela "MAC - Usuários" contém os endereços MAC de cada usuário, associados a identificadores (IDs). Esses IDs são usados na otimização (modelagem do problema), mais especificamente na identificação dos usuários e dos nós. Numa tabela denominada "DPIDs", são também associados aos IDs os DPIDs dos *switches*. O DPID refere-se a um identificador único do *switch* (DOVER, 2014) que o POX cria, ou seja, corresponde ao endereço do dispositivo de repasse. A "Tabela de Portas" contém informações sobre qual porta física é utilizada em um *switch* para se conectar a outro. Já a "Matriz de Custo" contém a latência de um *link* entre dois nós, ou seja, informa também se os nós não são vizinhos.

Quando uma configuração da rede é realizada, a aplicação de rede verifica os valores das variáveis contidas em "Variáveis da rede". Estas são variáveis utilizadas somente pelo

otimizador e não são descritas nos "Parâmetros da rede". Estas variáveis são descritas a seguir. "Sk" refere-se ao nó de entrada do usuário na rede, ou seja, ao primeiro *switch* pelo qual passa o fluxo do usuário. "Lk" é a latência do usuário até o nó "Sk" e "Ld" é a latência inserida anteriormente entre o usuário e Sk (caso seja a primeira otimização da rede, o seu valor é nulo). A latência "Ld" é subtraida de "Lk", sendo essa diferença informada ao controlador; caso contrário, a latência de entrada do usuário tenderia somente a aumentar.

Quando ocorre a chamada ao otimizador, o CPLEX é quem assume. São fornecidas como entrada do CPLEX as variáveis da rede e as tabelas de "Parâmetros da rede", porém antes de serem enviados, as entradas são convertidas para os IDs respectivos. O CPLEX gera duas saídas: as latências a serem adicionadas, que serão enviadas para cada usuário através do socket de mensagem e os caminhos dos fluxos que serão adicionados na tabela "Caminhos Salvos".

O "Controle de Fluxo" é responsável pela ação que será realizada no repasse dos pacotes. Caso a rede já esteja configurada, o "Controle de Fluxo" utiliza os "Caminhos Salvos" armazenados para determinar o próximo nó e então, através da "Tabela de Portas", é determinada a porta física que deve ser utilizada. Se a rede não estiver configurada são seguidas as regras das "Ações Predefinidas" que são as ações que devem ser tomadas durante a configuração, como o *Ping* entre os nós vizinhos, a ser apresentado na Seção 3.2. Depois de definida, a ação é repassada para o POX para que seja instalada a regra na tabela de fluxos.

3.2 **Funcionamento da aplicação de rede**

Esta seção aborda como a aplicação de rede atua desde a descoberta da rede e como as entidades representadas na Figura 6 interagem entre si.

Para a descoberta da rede é primeiramente feito um reconhecimento dos endereços: dos usuários e do servidor. Os endereços dos usuários e do servidor são identificados através das trocas de mensagens dos *sockets* e salvos nos "parâmetros da rede" e também através dos endereços de origem e DPID dos *packet ins* que são analisados pelo "Controle de Fluxo". Os usuários são identificados à medida que se conectam. Esses endereços são armazenados em tabelas e a eles são atribuídos IDs, ou seja, são identificados numericamente, pois pela modelagem apresentada no Capítulo 2 os nós seguem essa identificação.

Durante a fase de reconhecimento da rede, representada na Figura 5, é realizada a coleta dos parâmetros a serem utilizados na modelagem do problema, que são: as latências dos links, a latência entre o usuário e o *switch* ao qual ele se conecta e os nós nos quais o servidor e os usuários estão conectados.

A identificação dos nós é feita quando um *packet in* é analisado pelo "Controle

de Fluxo". Caso seja o primeiro pacote do endereço de origem analisado, significa que o *switch* que gerou o *packet in* é aquele ao qual o usuário ou o servidor está conectado. Então verifica-se o identificador do *switch* (DPID) e é realizada a referência ao ID.

Para a coleta das latências entre cada *switch*, são executados *pings* para todos os outros *switches*. O dispositivo de repasse com a tabela de fluxo desconfigurada gera um *packet in*. Como a rede ainda não está configurada, ainda são realizadas as "ações predefinidas", que são ações de configuração. A aplicação de rede então identifica no *packet in*, através do valor do DPID, em qual *switch* o pacote se encontra. Identifica-se então a origem do pacote e é realizada uma ação de *flooding*, para que seja enviado para os vizinhos. Esta lógica de ações garante que cada pacote de *ping* de cada *switch* somente será entregue para um nó vizinho. O mesmo processo é realizado entre os usuários e o *switch* ao qual o usuário está conectado.

O *ping* de medição é executado três vezes, pois o primeiro possui uma latência mais elevada devido à configuração do fluxo e então o valor médio obtido é enviado em uma mensagem pelo *socket*. Através dos *pings* para medição das latências dos *links* é então criada uma matriz com o valor latência, no qual o número da linha é o ID do ponto de origem e a coluna corresponde ao ID de destino, criando-se então uma matriz de custos (latências). Quando o *ping* de um *switch* não consegue alcançar outro *switch*, significa que os elementos não são vizinhos e o custo é preenchido com o valor zero na matriz de custos.

Durante todo o processo de reconhecimento da rede, sempre que um *packet in* é analisado, o número da porta de entrada é armazenado numa tabela, criando a tabela de portas apresentada na Figura 6.

Depois do reconhecimento inicial, a aplicação de rede possui todos os parâmetros da rede. Então é realizada uma chamada ao CPLEX, informando para ele as variáveis definidas no Capítulo 2 que foram armazenadas nas tabelas.

O CPLEX, por sua vez, informa para a aplicação de rede quais caminhos serão utilizados e qual a latência que deve ser adicionada a cada usuário. A latência é informada para o usuário através do *socket* e o usuário então adiciona a latência através do comando TC do Linux (TC, 2017). As latências adicionadas são armazenadas em uma variável (Ld) na aplicação de rede. Esta variável é utilizada quando há uma reconfiguração, pois a nova medição da latência entre o usuário e o primeiro nó estará acrescida da latência inserida. Logo, quando essa medição é feita, subtrai-se o valor inserido anteriormente. Caso contrário, pode haver alguma instabilidade na medição resultando em uma adição alta de latência para um usuário; e esta adição permaneceria durante toda a execução da aplicação, ou pode fazer com que a latência dos usuários aumente com o tempo, pois sempre seria adicionada latência e nunca subtraída.

A saída do CPLEX indica quais *links* serão utilizados por cada usuário. Assim, monta-se um vetor com o caminho que o fluxo de cada usuário deve seguir. Esse vetor

é então salvo na aplicação de rede ("Caminhos Salvos" na Figura 6) e o *flag* "Rede configurada" é *setado*, indicando que quando um usuário tentar se comunicar com o servidor, será seguido o caminho definido pelo otimizador CPLEX.

A reconfiguração da rede pode ser feita em paralelo com as outras funções da aplicação de rede. Para esta reconfiguração os fluxos das "ações predefinidas" já estão instalados nas tabelas de fluxos dos *switches*, fazendo com que a reconfiguração seja mais rápida do que na primeira vez.

Após a rede configurada, quando um *packet in* é gerado e a aplicação de rede verifica os endereços de origem e de destino. Caso os endereços pertençam a um usuário e ao servidor, o fluxo irá seguir o caminho fornecido pelo CPLEX. Definido o usuário, através do endereço de origem do *packet in*, se identifica o caminho que o fluxo deste usuário deve seguir, e o DPID indica qual o *switch* que fez a chamada. Através dos "caminhos salvos" define-se o *switch* no qual o fluxo deve ser repassado. Como o repasse do POX é dado através da porta física, verifica-se então na tabela de portas por qual porta física o fluxo deve ser enviado e é instalado então na tabela de fluxos do *switch* em que foi gerado o *packet in*. A ação de repasse também é instalada com um tempo de expiração (*timeout*). Esse parâmetro indica por quanto tempo a regra ficará instalada na tabela de fluxos. Como visto no Capítulo 1, todos os pacotes de um mesmo fluxos após o primeiro seguirão as regras já implementadas.

Quando for realizada uma reconfiguração, por parte da aplicação de rede e houver uma nova otimização, a aplicação de rede informa para os usuários a nova latência a ser inserida e redefine nos "Caminhos Salvos" a nova configuração dos fluxos. Porém os fluxos dos usuários não são atualizados imediatamente, sempre que ocorrer um *timeout* em um *switch* para qualquer configuração, será instalada a mais nova configuração.

4 AVALIAÇÃO DE DESEMPENHO

Para avaliar o desempenho da aplicação de rede proposta, foram utilizadas duas redes. A primeira é uma rede genérica que possui cinco nós, utilizada para comparar a aplicação proposta com dois modos básicos já implementados no controlador *POX*, os modos hub e *layer 2 learning*. A outra rede utilizada foi a rede acadêmica da RNP (RNP, 2016). Nesta topologia não foi utilizado o modo *hub*, pois esse não é capaz de realizar o repasse completo dos fluxos, dada a grande quantidade de nós na rede que acarreta em um número elevado de mensagens, sobrecarregando os recursos da rede.

Para as emulações foi utilizado o *Mininet*; um sistema que permite emulação rápida de grandes redes em um único computador. Ele cria redes definidas por software escaláveis usando mecanismos de virtualização leves. O *Mininet* possue recursos que criam, interajam, personalizam e compartilham os protótipos de rede criados rapidamente (OLIVEIRA et al., 2014). O *Mininet* utiliza protocolos e trocas de mensagens no formato real, fazendo com que a emulação seja bem próxima da operação de uma rede SDN. Em ambos os testes, para as redes de cinco nós e da RNP, utilizou-se o *Mininet* (MININET, 2017) para emular os dispositivos de repasse SDN e os usuários. Como apresentado na Figura 7, a aplicação de rede proposta interage com o CPLEX, fornecendo os parâmetros da rede e aplicando os caminhos calculados. Tanto a aplicação proposta como o CPLEX são executados dentro do controlador Pox. No caso dos outros dois modos usados na comparação (hub e layer 2 learning), ambos também são executados pelo *Pox*.

As emulações e a execução do controlador foram realizadas em uma máquina virtual, com o software *VMware Workstation 12* (VMWARE, 2012), com sistema operacional Ubuntu, memória RAM de 4 GB e dois núcleos do processador Intel Core i7-3770 com 3,40 GHz.

As medições de RTT (indiretamente medindo-se a latência) nos testes foram realizadas através da aplicação *Ping*. Como apresentado no Capítulo 3, é necessário que

Figura 7 - Esquema dos testes.

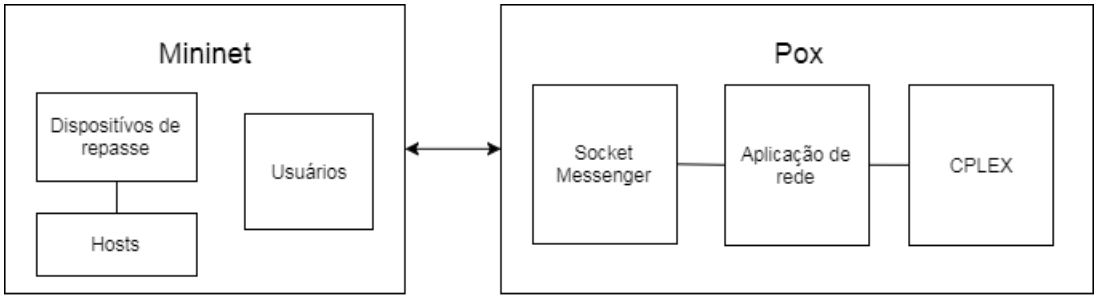
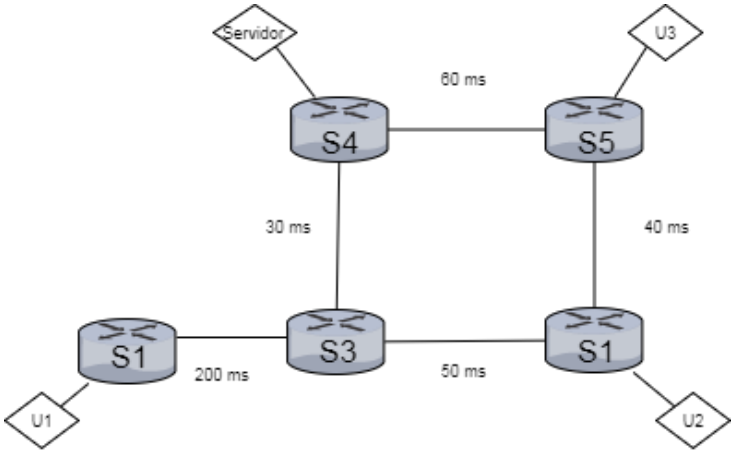


Figura 8 - Topologia da rede com cinco nós.



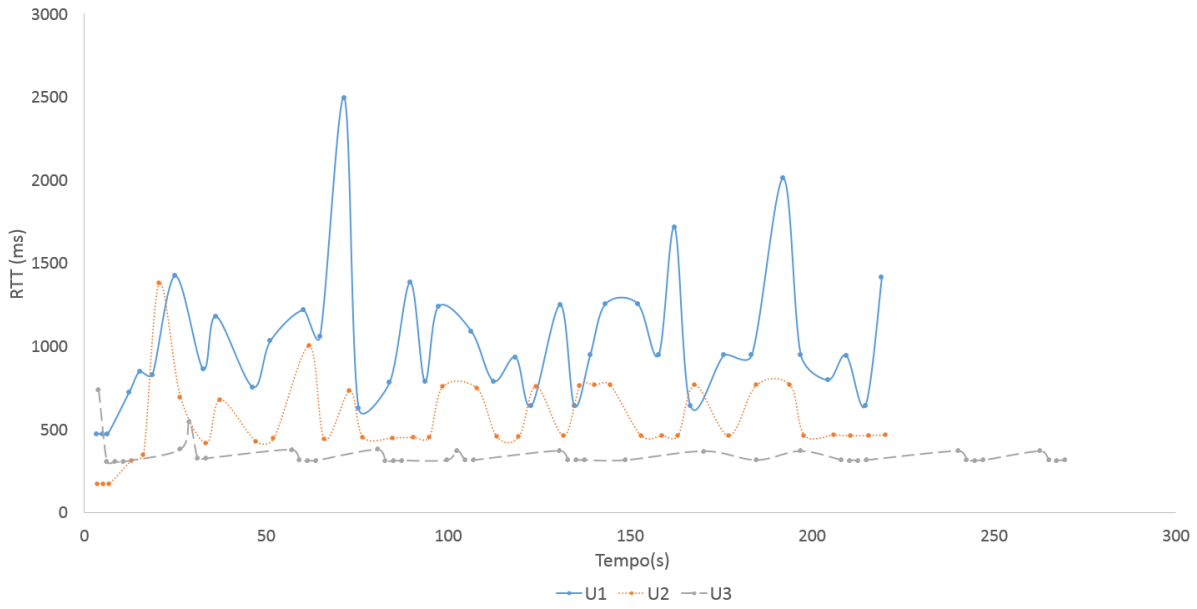
haja múltiplas reconfigurações da rede ao longo do tempo, em função das modificações das condições no estado da rede devidas ao tráfego. Assim, foram realizadas dez reconfigurações (iterações) em todos os casos, cujo o mecanismo proposto foi utilizado. Para cada reconfiguração da rede, foram feitas cinco medições com o objetivo de garantir a medição do fluxo já instalado, totalização cinquenta medições. Para os testes com o controlador nos outros modos (*hub* e *layer 2 learning* com *spanning tree*), como não há reconhecimento da rede, então não há número de iterações, foram utilizados o mesmo número de medições ao longo do tempo, para comparação com o método proposto. Cada teste foi realizado dez experimentos, afim de obter a média dos RTTs médios dos usuários e foi utilizado um intervalo de confiança de noventa por cento.

4.1 **Cenário 1: rede de cinco nós**

Para o teste de cinco nós foi utilizada a topologia representada na Figura 8, com três usuários em nós diferentes. De modo a avaliar a proposta em um caso simples inicialmente sem justiça, no link entre S1 e S3 foi introduzida uma latência elevada se comparada com a dos demais *links*. Esta situação força que o usuário U1 tenha uma latência maior do que a dos demais usuários. Não foi utilizado tráfego de fundo na rede de cinco nós, há somente os *pings* de testes.

Para a topologia referente à Figura 8, foram utilizadas no controlador cada uma das três aplicações consideradas: *hub*, *layer 2 learning* e com a justiça aplicada (proposta). O modo *hub* é o modo mais simples, pois há somente uma ação a ser tomada: quando um *packet in* é gerado, o controlador realiza a ação de *flooding*, ou seja, envia o pacote para todas as portas físicas com exceção da porta pela qual o pacote foi recebido. O outro

Figura 9 - RTT em função do tempo para o modo *hub* – rede de cinco nós.



modo, *layer 2 learning*, é um método simples de repasse de pacotes que cria uma tabela na aplicação de rede com todos os endereços fonte dos pacotes que chegam no *switch* e suas respectivas portas. Assim, sempre que um pacote com um determinado destino estiver na tabela, o *switch* repassa o pacote para a porta correspondente. Caso contrário, ação é realizado *flooding* do pacote.

As Figuras 9, 10 e 11 apresentam o comportamento do RTT em função do tempo de apenas uma das amostras do experimento, para cada uma das três aplicações. Nesses gráficos, o eixo horizontal representa o tempo e o eixo vertical representa o RTT de cada usuário.

No modo *hub* (Figura 9), nota-se que o RTT varia muito em função do tempo. Como não há decisões inteligentes, pois em cada dispositivo de repasse todo pacote é repassado para todas as portas (com exceção da porta de entrada), devido aos *loops* criados, há muito tráfego, o que gera grande variação no RTT. Por exemplo, para o usuário U1, o RTT é bem maior do que o dobro da menor latência até o servidor (472 ms). Convém destacar que, também como esperado, o maior RTT é obtido para o usuário U1.

No modo *layer 2 learning* (Figura 10), foi utilizado juntamente o protocolo *spanning tree* de forma a eliminar os *loops* na rede. Dessa forma, obteve-se uma latência menor e mais estável quando comparada à obtida no modo *hub*. Os picos de latência ocorrem devido a *timeouts*, ou seja, ao tempo em que uma regra fica instalada na tabela de fluxo do *switch*; isto é, o tempo de validade da regra. No caso de um *timeout*, há necessidade de reconfiguração do *switch*, levando a necessidade de obtenção da uma nova regra (que pode até ser a mesma anterior). Para isso, há a geração de *packet ins* e suas

Figura 10 - RTT em função do tempo para o modo *layer 2 learning* – rede de cinco nós.

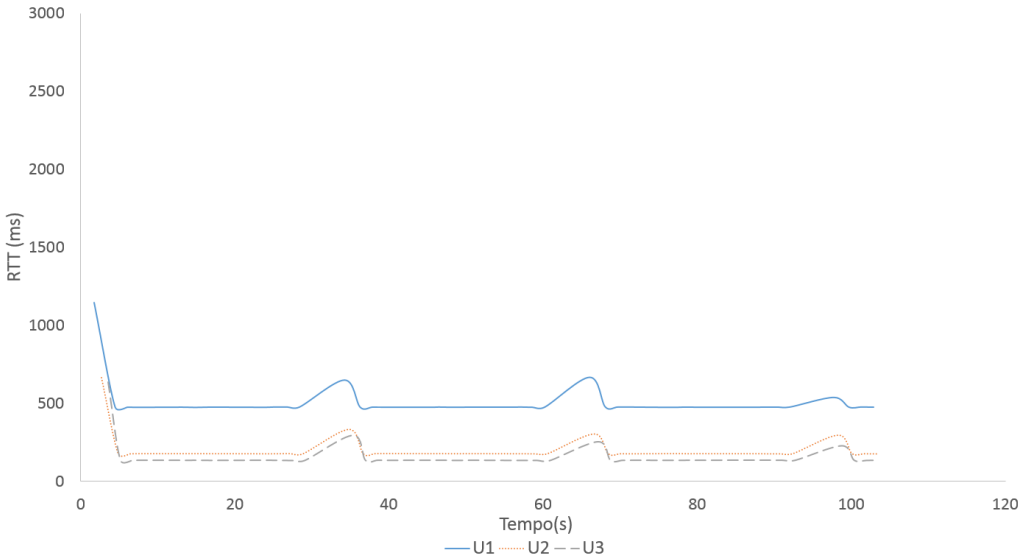
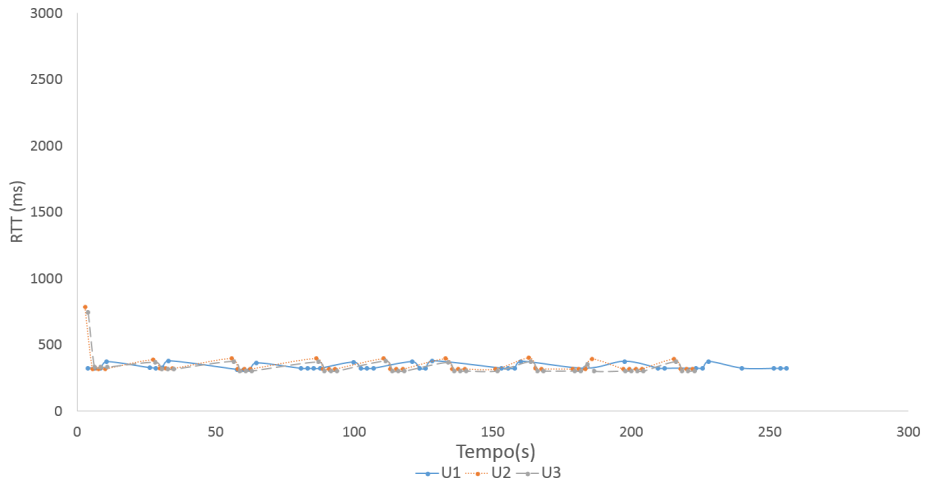


Figura 11 - RTT em função do tempo para o modo justiça – rede de cinco nós.



respectivas respostas, o que eleva o tempo de resposta. A obtenção da regra é o que causa a latência maior no primeiro pacote; o controlador precisa de tempo para aplicar a regra corretamente, levando o primeiro RTT a ser muito grande. Pode-se observar que os RTTs se aproximam dos valores relativos ao dobro das menores latências até o servidor.

No modo com justiça ativa (Figura 11), ocorre o mesmo comportamento referente à primeira latência. Porém, pode-se observar uma menor diferença entre os RTTs dos usuários para o servidor, o que indica a eficácia da solução proposta.

Diferentemente da Figura 10, na qual cada RTT se encontra em um próprio patamar e é constante na maioria do tempo, na Figura 11, todas os RTTs medidos se encontram mais próximos, porém com um determinado acréscimo para os usuários U2 e U3 e com uma oscilação para todos os usuários, causados pela aplicação de rede que tentar equalizar as latências. Convém lembrar que quando um dos usuários possui um maior RTT, a aplicação de rede pode adicionar atraso para os outros usuários, de modo que haja uma maior justiça. Além disso, novamente os picos ocorrem devido a *timeouts* das regras.

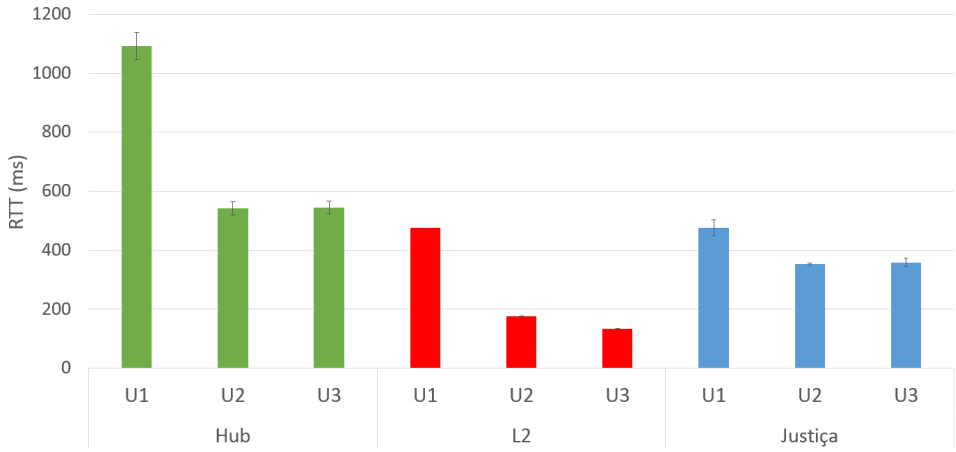
A Figura 12 apresenta a média do RTT médio de cada usuário em cada modo, de forma a permitir uma melhor comparação entre os modos de operação. Para cada experimento foi calculado o RTT médio (média do RTT no tempo). Em seguida, foi calculada a média desses RTTs médios considerando os dez experimentos. Comparando-se os RTTs entre o controlador nos modos *hub*, *layer 2 learning* (L2 na Figura 12) e com o modo justiça (representado por Ativo), nota-se que o RTT no modo *hub* é maior do que para os outros dois casos, o que já era esperado. Porém, comparando-se os resultados do usuário U1 para os modos *layer 2 learning* e justiça, percebe-se que o RTT é aproximadamente igual. Entretanto, no caso da aplicação de rede proposta, os RTTs dos outros usuários estão mais próximos do RTT do usuário U1, indicando então a eficácia do mecanismo proposto.

4.2 **Cenário 2: rede da RNP**

A Figura 13 mostra a topologia da rede da RNP utilizada nas avaliações a seguir. A RNP é uma instituição que gerencia uma rede acadêmica de 27 nós e 33 *links* (RNP, 2016). Foram utilizadas as mesmas topologia, banda e latência de (COUTO et al., 2015).

A RNP possui pontos de presença (*Points of Presence* – PoPs) espalhados por diversas cidades do Brasil. Para escolher o posicionamento de cada usuário na avaliação realizada, foram consideradas as capitais com maior população do Brasil, com exceção de Brasília que foi escolhida como nó servidor, por ser capital do país e por estar situada em uma posição central. Foram realizados testes com dois, quatro e oito usuários, com o objetivo de avaliar como o modo proposto se comporta para diferentes números de

Figura 12 - Média do RTT médio – rede de cinco nós.



usuários, como competições de jogos são duas equipes, foram utilizados números pares de usuários. Logo, para o caso de dois usuários, um estará conectado a um *switch* posicionado em São Paulo e outro no Rio de Janeiro; para o caso de quatro usuários também serão utilizados *switches* em Salvador e Fortaleza; por último, para o oito usuários, também são utilizados *switches* em Belo Horizonte, Manaus, Curitiba e Recife.

Os testes foram realizados com os mesmos modos utilizados na topologia com cinco nós, com a exceção do modo *hub*, pois para uma rede com o tamanho da rede da RNP, o *flooding* produz um desempenho muito ruim. Além disso, foram adicionados tráfegos TCP entre cada usuário e o servidor para simular tráfego da aplicação para a qual a proposta procura diminuir as diferenças entre as latências dos usuários para o servidor. Para isso foi utilizada a ferramenta Iperf. Da mesma forma que no cenário anterior, foram utilizados *Pings* para medir o RTT.

A Figura 14 apresenta o RTT em função do tempo para uma execução do modo *layer 2 learning* e a Figura 15 apresenta o mesmo para o modo proposto. Para o caso da execução do controlador no modo *layer 2 learning*, é possível perceber uma grande variação do RTT. Além disso, os valores dos RTTs dos dois usuários para o servidor estão bem diferentes. Na Figura 15, na qual o mecanismo proposto é utilizado, é possível observar que o RTT dos usuários está mais estável. Isto é obtido através do cálculo proativo dos caminhos entre os usuários e servidor, logo a aplicação de rede não precisa calcular os caminhos à medida que fluxos são gerados. Percebe-se também que o RTT entre os dois usuários e o servidor no modo proposto estão mais próximos. Na mesma figura, os picos são causados principalmente pelos *timeouts* que disparam as reconfigurações. No tempo de 200 ms há somente um pico de RTT entre os usuários, é provável que a reconfiguração do fluxo provavelmente foi realizada através do fluxo TCP e quando foi executado o *ping* de medição o fluxo já estava configurado.

Figura 13 - Topologia da rede da RNP.

Fonte: Adaptado de RNP, 2016.

Figura 14 - RTT em função do tempo para o modo *layer 2 learning* – rede da RNP com dois usuários.

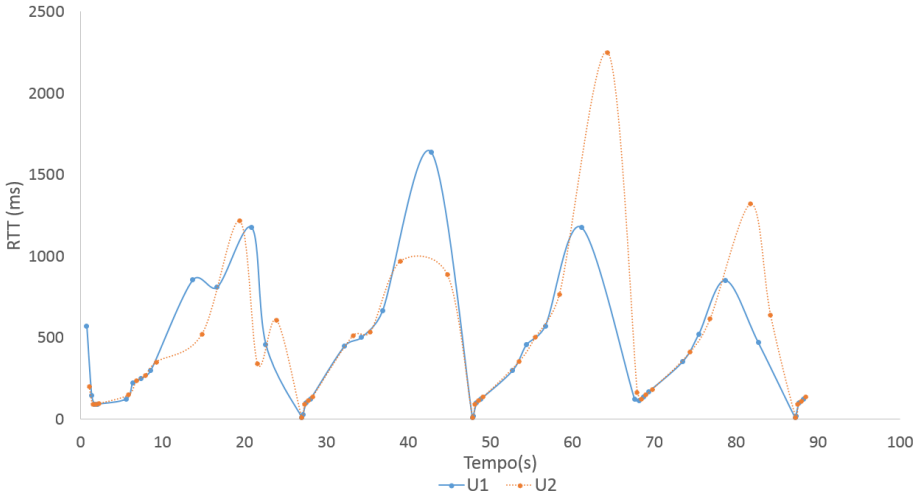


Figura 15 - RTT em função do tempo para modo justiça – rede da RNP com dois usuários.

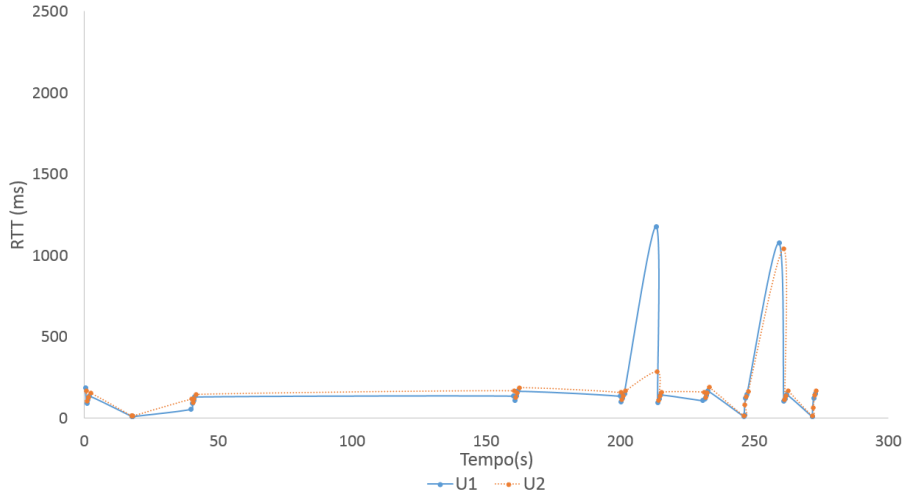
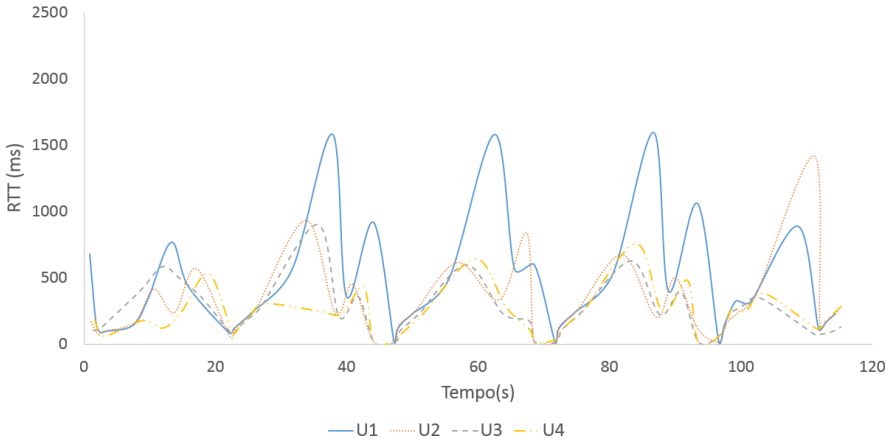


Figura 16 - RTT em função do tempo para o modo *layer 2 learning* – rede da RNP com quatro usuários.



As Figuras 16 e 17 apresentam o RTT em função do tempo para uma execução para quatro usuários. Como no caso anterior, os RTTs dos usuários para o servidor no modo *layer 2 learning* variam consideravelmente, enquanto que no mecanismo proposto os RTTs dos usuários se encontram bem próximos.

Também foram realizados experimentos para oito usuários, mas seus comportamentos no tempo são de difícil visualização. Dessa forma, esses resultados foram omitidos e serão analisadas apenas as médias dos RTTs médios, apresentadas a seguir.

As Figuras 18, 19 e 20 apresentam a média do RTT médio de cada usuário com o controlador em ambos os modos.

Para o caso com dois usuários, os RTTs de ambos os usuários ainda estão bem próximos no método proposto, embora menos próximos do que no cenário anterior. No cenário utilizado, mesmo sem a aplicação da proposta, os dois usuários possuem RTTs para o servidor bem próximos (a diferença é de 36 ms), o que levaria à conclusão de que a proposta não é tão necessária. Porém, no método proposto os usuários possuem RTTs menores do que no modo *layer 2 learning*, pois os caminhos dos usuários são calculados proativamente, levando-se em conta toda a rede. Nesta situação é possível perceber uma melhoria significativa nos RTTs dos usuários. Isso ocorre, pois a otimização do CPLEX, mesmo com possível adição artificial de atraso, encontra soluções mais eficientes do que as encontradas pelo *layer 2 learning* com *spanning tree*.

Na Figura 19, na qual o número de usuários aumenta para quatro, é possível perceber que os RTTs dos usuários também aumentam a ponto de estarem perto dos RTTs do modo *layer 2 learning*. Isto é causado pelo fato de pelo menos um dos usuários possuir um RTT maior do que o dos demais fazendo com que o controlador aumente os RTT dos outros usuários para criar um cenário mais justo. Comparando-se as Figuras 18 e 19, nota-se para o método proposto um aumento nos RTTs dos usuários U1 e U2 devido

Figura 17 - RTT em função do tempo para modo justiça – rede da RNP com quatro usuários.

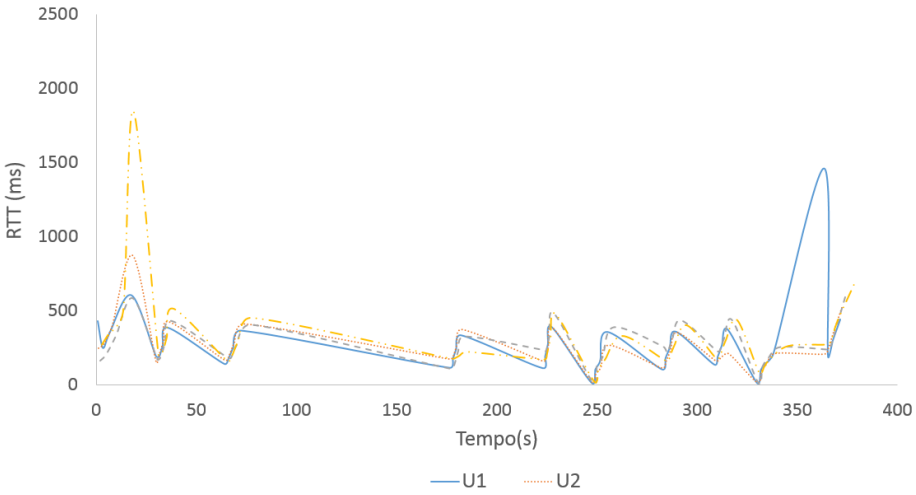


Figura 18 - Média do RTT médio – rede da RNP com dois usuários.

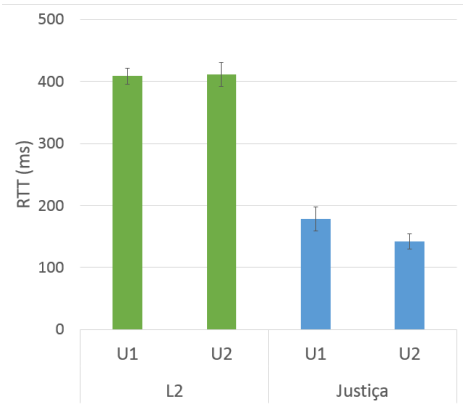


Figura 19 - Média do RTT médio – rede da RNP com quatro usuários.

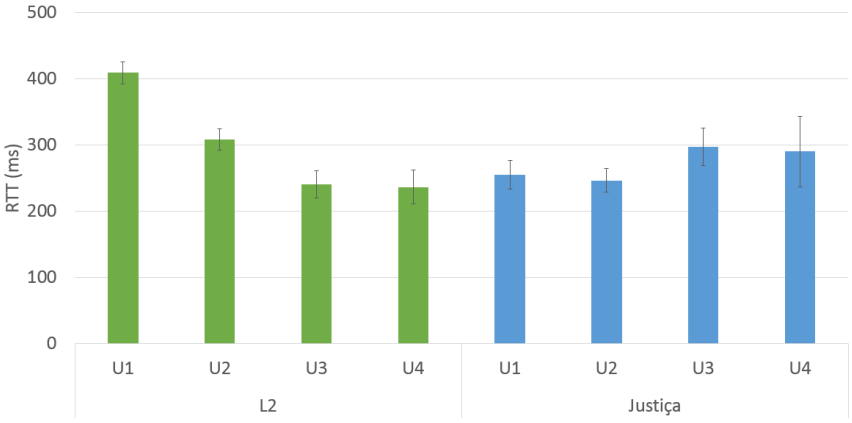
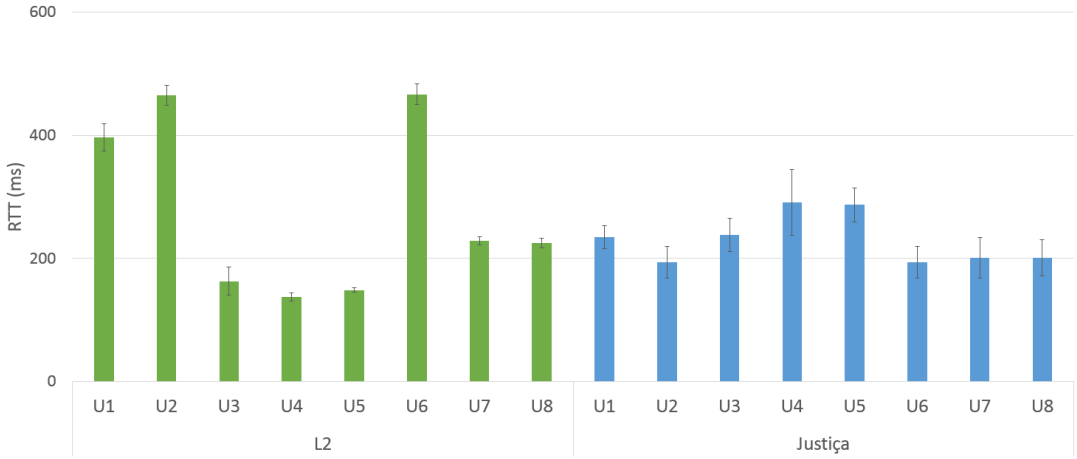


Figura 20 - Média do RTT médio – rede da RNP com oito usuários.



à inserção dos usuários U3 e U4.

Na Figura 20, na qual o número de usuários é igual a oito, novamente o modo proposto consegue alcançar uma maior justiça entre os usuários.

5 CONCLUSÃO

Esta dissertação propõe uma aplicação de rede que calcula os caminhos dos usuários para um único servidor provendo a maior justiça entre todos os usuários enquanto tenta obter a menor latência possível sem prejudicar a justiça. Na solução proposta, redes definidas por software foram utilizadas, com o controlador POX, e modelou-se um problema de programação linear, solucionado pelo otimizador CPLEX.

Através da aplicação de rede proposta foi possível implementar com uma linguagem de alto nível, com um controlador centralizado, através da arquitetura SDN. Esta aplicação de redes é capaz de coletar dados dos usuários e armazená-los de maneira centralizada para que, utilizando essas informações, o otimizador ache soluções para o conjunto de usuários devido à centralização do controle. A análise da ocupação da rede com a visão completa da mesma e a capacidade do controlador de manipular os fluxos de todos os usuários ao mesmo tempo facilitam o controle da QoS, dando uma maior justiça aos usuários.

A partir dos resultados foi possível analisar o comportamento da latência dos usuários ao longo do tempo. Percebe-se que a atuação da aplicação proposta consegue diminuir as diferenças entre as latências de cada usuário para o servidor, tornando essas latências próximas. Em comparação com os modos do POX *hub* e *layer 2 learning* avaliados, cada usuário tem latência próxima a dos outros usuários; porém nem sempre menor do que as latências no modo *layer 2 learning*.

A aplicação também se mostrou capaz de ser mais eficiente do que o modo *layer 2 learning* com *spanning tree*, dependendo das condições; por exemplo, se um usuário possuir uma latência relativamente alta, a aplicação de rede nesse caso pode induzir uma latência para um usuário maior do que seria obtida em outros métodos, com o objetivo de obter justiça. Pode-se notar através dos resultados, que independentemente do número de usuários, é possível obter um cenário mais justo. Principalmente para redes com uma quantidade mais elevada de nós, ter uma visão geral da rede traz vantagens, pois é possível otimizar um fluxo, adaptando-se à medida que eventos vão acontecendo.

Para trabalhos futuros é possível realizar modificações no código para um melhor desempenho. Por exemplo, é possível implementar o parâmetro de banda na modelagem do problema. Isso permite que perdas de pacotes por congestionamento sejam evitadas e reduzir variações no RTT para certos usuários.

Outro trabalho possível corresponde a realizar com uma aplicação interativa na qual existam seções críticas, ou seja, na qual há momentos em que a informação em uma aplicação é mais sensível à qualidade de serviço (DEBROY et al., 2013). É possível adicionar uma aplicação de rede com a capacidade de priorizar certos fluxos, pois as aplicações de rede podem se comunicar e prever as seções críticas melhorando assim a

justiça entre os usuários.

REFERÊNCIAS

ACHTERBERG, T.; BERTHOLD, T. Improving the feasibility pump. *Discrete Optimization*, Elsevier, v. 4, n. 1, p. 77–86, 2007.

AZODOLMOLKY, S. *Software Defined Networking with OpenFlow*. [S.l.]: Packt Publishing Ltd, 2013.

BHAT, S. et al. *Can SDN controller based NSCs help improve user experience of online games?* [S.l.], 2015.

BRUN, J.; SAFAEI, F.; BOUSTEAD, P. Managing latency and fairness in networked games. *Communications of the ACM*, ACM, v. 49, n. 11, p. 46–51, 2006.

CHEN, K.-T. et al. On the sensitivity of online game playing time to network qos. In: *INFOCOM*. [S.l.: s.n.], 2006.

COUTO, R. S. et al. GT-PID: Uma nuvem IaaS universitária geograficamente distribuída. In: REDCLARA. *Quinta Conferencia de Directores de Tecnología de Información - TICAL 2015*. [S.l.], 2015. p. 1–19.

DEBROY, S. et al. Critical sections in networked games. In: IEEE. *Communications (ICC), 2013 IEEE International Conference on*. [S.l.], 2013. p. 2365–2369.

DOVER, J. *A switch table vulnerability in the Open Floodlight SDN controller*. [S.l.]: Dover Networks LLC, Tech. Rep, 2014.

GITHUB: Github - noxrepo/pox. 2017. Disponível em: ⟨https://github.com/noxrepo/pox⟩. Acesso em: 22 jul. 2017.

GOLDBARG, M. C.; LUNA, H. P. L. *Otimização combinatória e programação linear: modelos e algoritmos*. [S.l.]: Elsevier, 2005.

GORLATCH, S.; HUMERNBRUM, T. Enabling high-level qos metrics for interactive online applications using sdn. In: IEEE. *Computing, Networking and Communications (ICNC), 2015 International Conference on*. [S.l.], 2015. p. 707–711.

HENDERSON, T.; BHATTI, S. Networked games: a qos-sensitive application for qos-insensitive users? In: ACM. *Proceedings of the ACM SIGCOMM workshop on Revisiting IP QoS: What have we learned, why do we care?* [S.l.], 2003. p. 141–147.

HINRICHS, T. L. et al. Practical declarative network management. In: ACM. *Proceedings of the 1st ACM workshop on Research on enterprise networking*. [S.l.], 2009. p. 1–10.

HUANG, S.; GRIFFIOEN, J. Hypernet games: Leveraging sdn networks to improve multiplayer online games. In: IEEE. *Computer Games: AI, Animation, Mobile, Interactive Multimedia, Educational & Serious Games (CGAMES), 2013 18th International Conference on*. [S.l.], 2013. p. 74–78.

HUANG, S.; GRIFFIOEN, J.; CALVERT, K. L. Network hypervisors: Enhancing sdn infrastructure. *Computer Communications*, Elsevier, v. 46, p. 87–96, 2014.

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, 2014.

JAIN, R.; PAUL, S. Network virtualization and software defined networking for cloud computing: a survey. *IEEE Communications Magazine*, IEEE, v. 51, n. 11, p. 24–31, 2013.

JAMJOOM, H.; WILLIAMS, D.; SHARMA, U. Don't call them middleboxes, call them middlepipes. In: ACM. *Proceedings of the third workshop on Hot topics in software defined networking*. [S.l.], 2014. p. 19–24.

JIN, D. Y.; CHEE, F. Age of new media empires: A critical interpretation of the korean online game industry. *Games and culture*, Sage Publications Sage CA: Los Angeles, CA, v. 3, n. 1, p. 38–58, 2008.

KOHAVI, R.; LONGBOTHAM, R. Online experiments: Lessons learned. *Computer*, IEEE, v. 40, n. 9, 2007.

KREUTZ, D. et al. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, IEEE, v. 103, n. 1, p. 14–76, 2015.

LI, W. et al. Joint optimization of bandwidth for provider and delay for user in software defined data centers. *IEEE Transactions on Cloud Computing*, IEEE, 2015.

MAXEMCHUK, N. F.; SHUR, D. H. et al. An internet multicast system for the stock market. *ACM transactions on computer systems*, v. 19, n. 3, p. 384–412, 2001.

MININET: Instant virtual network. 2017. Disponível em: ⟨http://mininet.org/⟩. Acesso em: 22 jul. 2017.

OLIVEIRA, R. L. S. D. et al. Using mininet for emulation and prototyping software-defined networks. In: IEEE. *Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on*. [S.l.], 2014. p. 1–6.

RNP: Rede ipê. 2016. Disponível em: ⟨https://www.rnp.br/servicos/conectividade/rede-ipe⟩. Acesso em: 8 fev. 2017.

SHARMA, S. et al. Demonstrating resilient quality of service in software defined networking. In: IEEE. *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*. [S.l.], 2014. p. 133–134.

TC: Traffic control in the linux kernel. 2017. Disponível em: ⟨http://lartc.org/manpages/tc.txt⟩. Acesso em: 8 ago. 2017.

VMWARE: Máquina virtual. 2012. Disponível em: ⟨https://www.vmware.com/⟩. Acesso em: 22 jul. 2017.