Utilizando Redes Definidas por Software para Prover Justiça em Aplicações Interativas

Felipe A. F. da Silva, Rodrigo S. Couto, Marcelo G. Rubinstein *

[1]Universidade do Estado do Rio de Janeiro - FEN/PEL/DETEL

{farrudafernandes}@gmail.com,{rodrigo.couto,rubi}@uerj.br

Resumo. *Aplicações interativas e em tempo real, como jogos online, geralmente têm requisitos estritos de latência. Mais especificamente, podem necessitar que os usuários possuam valores similares de latência; caso contrário, pode haver injustiça entre eles. Este trabalho propõe uma aplicação para redes definidas por software que garante que as latências entre cada usuário e o servidor sejam as mais próximas possíveis. Essa aplicação consiste em resolver periodicamente um problema de otimização que leva em consideração a latência dos enlaces e calcula os caminhos dos fluxos na rede. A partir de experimentos realizados com emulação, nota-se que a aplicação de rede proposta consegue obter um nível de justiça maior do que o de uma rede tradicional.*

Abstract. *Interactive and real-time applications, such as online games, often have strict latency requirements. More specifically, they may require users to have similar latency values; otherwise there may be unfairness between them. This work proposes an application for software defined networks that ensures that the latencies between each user and the server are as close as possible. This application periodically solves an optimization problem that takes latency of the links into account and chooses the paths of the flows in the network. From the experiments carried out with emulation, it can be noticed that the proposed network application achieves a level of fairness greater than that of a traditional network.*

1. Introdução

Cada vez há mais aplicações utilizadas por um grande número de usuários e que, além disso, são sensíveis a requisitos de Qualidade de Serviço (QoS) [Gorlatch e Humernbrum, 2015]. Em uma aplicação interativa e em tempo real, um requisito crítico de QoS é a latência entre os usuários e o servidor da aplicação. Como os usuários desse tipo de serviço podem estar geograficamente distribuídos, esses são afetados por níveis diferentes de latência com o servidor. Nessas aplicações, possuir latências diferentes pode levar a visões diferentes de um mundo digital, criando cenários injustos. A justiça é importante, por exemplo, em jogos multiusuário em tempo real [Humernbrum et al., 2014] ou em algumas operações financeiras [Maxemchuk et al., 2001].

Em relação aos jogos multiusuários em tempo real, um requisito fundamental é a consistência; ou seja, todos os jogadores devem compartilhar uma mesma visão do

*Este trabalho foi realizado com recursos da FAPERJ, CNPq e CAPES e dos processos nº 15/24494-8 e nº 15/24490-2, da Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP).

mundo digital. Entretanto, se há latências diferentes entre o servidor e os jogadores, cada jogador tem uma visão diferente do estado do jogo. Certas aplicações realizam predição do movimento dos usuários caso haja falhas na rede. No entanto, 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 jogo de tiro em primeira pessoa pode ser seriamente afetado se o RTT (*Round-Trip Time* - tempo de ida e volta) for acima de 60 ms [Brun et al., 2006]. Assim, muitas vezes as competições oficiais de jogos multiusuários são presenciais, utilizando redes locais. Porém, com uma abordagem de maior justiça, é possível viabilizar competições em redes de longa distância.

Este trabalho propõe uma aplicação de rede capaz de minimizar as diferenças entre as latências de vários usuários para um único servidor. Para tal, considera-se que os usuários se conectam ao servidor por uma rede definida por *software* (*Software-Defined Network* - SDN) de longa distância. O emprego de SDN possibilita criar aplicações que aplicam 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. Assim, a aplicação proposta resolve periodicamente um problema de otimização que tenta igualar as latências dos usuários ao servidor, considerando a latência dos enlaces da rede. Como saída do problema, os caminhos dos fluxos na rede são calculados e implementados pelo controlador SDN. A aplicação é desenvolvida para o controlador POX [POX, 2017] e, para analisar seu desempenho, utiliza-se o emulador Mininet [Mininet, 2017]. Os resultados obtidos mostram que é possível atingir um nível de justiça maior do que o de uma rede convencional.

Este trabalho está organizado da seguinte forma. A Seção 2 apresenta os trabalhos relacionados. O modelo de otimização proposto é descrito na Seção 3, enquanto a Seção 4 descreve a implementação. Os resultados são apresentação na Seção 5. Finalmente, a Seção 6 conclui o trabalho e indica direções futuras.

2. Trabalhos Relacionados

Diversos trabalhos [Debroy et al., 2013, Brun et al., 2006] avaliam o desempenho de aplicações sensíveis a requisitos de QoS, principalmente de aplicações de jogos.

[Debroy et al., 2013] analisaram gravações de mais de cinco horas de vídeos de jogos de tiro em primeira pessoa envolvendo 10 voluntários e identificaram sessões nas quais a degradação da QoS causa inconsistência no estado do jogo, resultando em uma má experiência dos usuários. Os autores também mostram o quanto a latência e a variação dela interferem no desempenho dos usuários.

[Brun et al., 2006] apresentam 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.

Outros trabalhos [Gorlatch e Humernbrum, 2015, Huang e Griffioen, 2013, Li et al., 2017] buscam melhorar o desempenho e a justiça das aplicações sensíveis a requisitos de QoS usando redes SDN.

Em [Gorlatch e Humernbrum, 2015], são utilizados os recursos de gerenciamento dinâmico da rede SDN para expressar, monitorar e controlar as demandas de QoS de aplicações *online* e interativas de tempo real. Os autores desenvolveram uma API *north-bound* para atender aos requisitos de QoS dessas aplicações. Além disso, demonstrou-se como uma métrica de 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 em uma rede OpenFlow.

[Huang e Griffioen, 2013] propõem uma abstração denominada HyperNet Games que interage com um jogo, sendo capaz de criar e implantar dinamicamente uma rede SDN que se adapta às necessidades do jogo e aos seus participantes. Os autores também mostram como a aplicação ajuda na personalização da rede para jogos quando comparada com uma rede pública, como a Internet, e que o RTT de um usuário diminui consideravelmente.

Uma aplicação com controle centralizado utilizando SDN é apresentada em [Li et al., 2017]. Utilizando um modelo de otimização, essa aplicação determina a melhor configuração possível em termos de uso de banda e latência, tanto para o provedor de serviço quanto para o usuário. Com a solução proposta, fluxos são redirecionados para os servidores que estão sendo menos utilizados em um determinado momento.

Este trabalho, de forma similar à proposta de [Li et al., 2017], também propõe uma aplicação de otimização em SDN. Entretanto, este trabalho considera que a rede é bem provisionada e não otimiza o uso da banda, focando em melhorar a justiça em termos de latência. Além disso, [Li et al., 2017] trata da justiça de forma que o desvio padrão da latência entre os usuários seja reduzido. Por outro lado, neste trabalho o problema de otimização calcula as variáveis de decisão visando igualar a latência entre os usuários. Realizar otimização para igualar a latência é uma medida conservadora para melhorar a justiça, levando em consideração que pequenos desvios entre as latências podem ocorrer em um cenário prático. Assim, este trabalho mostra a atuação do problema de otimização de forma prática, por meio de emulação. Em contrapartida, [Li et al., 2017] apresenta apenas resultados teóricos.

3. Modelagem do Problema

Este trabalho considera que os usuários de uma aplicação interativa se conectam ao servidor por meio de uma rede SDN de longa distância. No Controlador SDN, executa-se um problema de otimização que calcula o caminho de cada usuário para o servidor, forçando que a latência entre os usuários sejam iguais. Por simplicidade e levando em consideração a geodistribuição, a latência é definida como o a soma dos atrasos de propagação dos enlaces no caminho entre cada usuário e o servidor. Como o requisito de os usuários possuírem caminhos com latência iguais pode levar à inexistência de solução para o problema, aplica-se um atraso compensatório em cada usuário. Esse atraso pode ser aplicado diretamente na aplicação do usuário ou por meio de filas nos comutadores.

O problema de otimização deste trabalho é modelado com um problema de programação linear inteira mista (*Mixed-Integer Linear Programming* - MILP). A Tabela 1 apresenta as notações utilizadas no modelo. Os parâmetros e conjuntos são os dados de entrada, enquanto as variáveis são os valores escolhidos na solução do problema. As

equações do MILP são apresentadas a seguir.

Tabela 1. Notações utilizadas no problema.

Notação	Descrição	Tipo
$\mathcal{S}$	Comutadores da rede SDN	Conjunto
$\mathcal{K}$	Usuários	Conjunto
L_{ij}	Latência entre os comutadores i e j	Parâmetro
A_k	Latência do enlace de acesso do usuário k até o nó de entrada na rede SDN	Parâmetro
$\mathcal{I}_k$	Comutador de entrada do usuário k na rede SDN	Parâmetro
$\mathcal{O}$	Comutador no qual o servidor está localizado	Parâmetro
e_{ijk}	Variável binária indicando se o usuário k utiliza o enlace entre os nós i e j	Variável
δ_k	Atraso compensatório para o usuário k	Variável
l	Latência dos usuários para o servidor	Variável

$$\text{minimizar } \; l \tag{1}$$

$$\text{sujeito a } \; l = A_k + \sum_{i,j \in S} L_{ij} e_{ijk} + \delta_k, \quad \forall k \in \mathcal{K} \tag{2}$$

$$\sum_{j \in D} e_{ijk} - \sum_{j \in D} e_{jik} = \begin{cases} 1 & \text{se } i = \mathcal{I}_k \text{ ,} \\ -1 & \text{se } i = \mathcal{O} \text{ ,} \\ 0 & \text{caso contrário.} \end{cases} \qquad \forall k \in \mathcal{K}, i \in \mathcal{S} \tag{3}$$

$$l \in \mathbb{R}^+; \quad \delta_k \in \mathbb{R}^+ \; \forall k \in \mathcal{K}; \quad e_{ijk} \in \{0,1\} \; \forall k \in \mathcal{K}, \; i,j \in \mathcal{S} \tag{4}$$

O objetivo do problema, dado pela Equação 1, é minimizar a latência de cada usuário para o servidor. Como o problema deve forçar que todos os usuários tenham o mesmo valor de latência, uma única variável l representa a latência a ser minimizada. A Equação 2 então calcula o valor de l, considerando a latência dos usuários. Assim, para cada usuário $k \in \mathcal{K}$, o lado direito da Equação 2 corresponde à soma do atraso do enlace de acesso do usuário à rede SDN, representado por A_k, com o atraso total do seu caminho até o servidor dentro da rede SDN, representado pelo somatório $\sum_{i,j \in S} L_{ij} e_{ijk}$, e com o atraso compensatório δ_k. O sinal de igualdade da Equação 2 garante que todos os usuários tenham o mesmo valor de latência l para o servidor. A Equação 3 é utilizada para conservação de fluxos; ou seja, essa equação garante que um caminho comece no comutador $\mathcal{I}_k$ ao qual o usuário se conecta e termine no comutador $\mathcal{O}$, que está conectado ao servidor. Assim, para um comutador $\mathcal{I}_k$, deve haver apenas um fluxo do usuário k saindo desse nó, enquanto para o comutador $\mathcal{O}$ deve haver apenas um fluxo do usuário k entrando nesse nó (isto é, somatório dos fluxos é -1). Para os demais comutadores, que são intermediários na rede, o somatório dos fluxos que entram em um nó deve ser igual ao somatório dos fluxos que saem desse nó, para um determinado usuário k. Finalmente, a Equação 4 determina o domínio das variáveis.

4. Implementação

O problema de otimização formulado visa oferecer maior justiça entre usuários em uma SDN, como visto na Seção 3. 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 pelo CPLEX [Achterberg e 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 desse otimizador. Neste trabalho essa aplicação é desenvolvida em Python no controlador POX [POX, 2017]. Utiliza-se o POX devido à sua simplicidade de instalação e programação. Entretanto, é possível implementar a proposta deste trabalho em outros controladores, como o OpenDaylight [Foundation, 2017]. A comunicação entre os componentes da solução proposta está representada na Figura 1. Essa figura mostra que a aplicação de rede desenvolvida tem a função de ser um módulo intermediário entre o controlador POX, que executa as ações, e o otimizador, no qual são calculados os caminhos dos fluxos e os atrasos compensatórios. A arquitetura e o funcionamento da aplicação desenvolvida são apresentados a seguir.

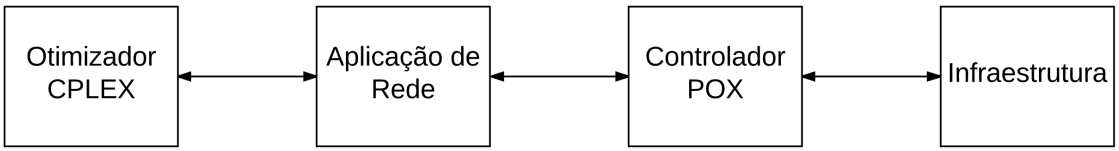


Figura 1. Trocas de informações entre os componentes da solução proposta.

4.1. Arquitetura da aplicação de rede

A Figura 2 apresenta o diagrama de blocos da aplicação de rede e como ela interage com as entidades externas, que são: os usuários, os comutadores, o controlador POX e otimizador CPLEX. A aplicação de rede interage diretamente com os usuários e os comutadores por meio do Encaminhador de Mensagens e indiretamente com os comutadores pelo POX, devido à abstração que esse controlador realiza da rede.

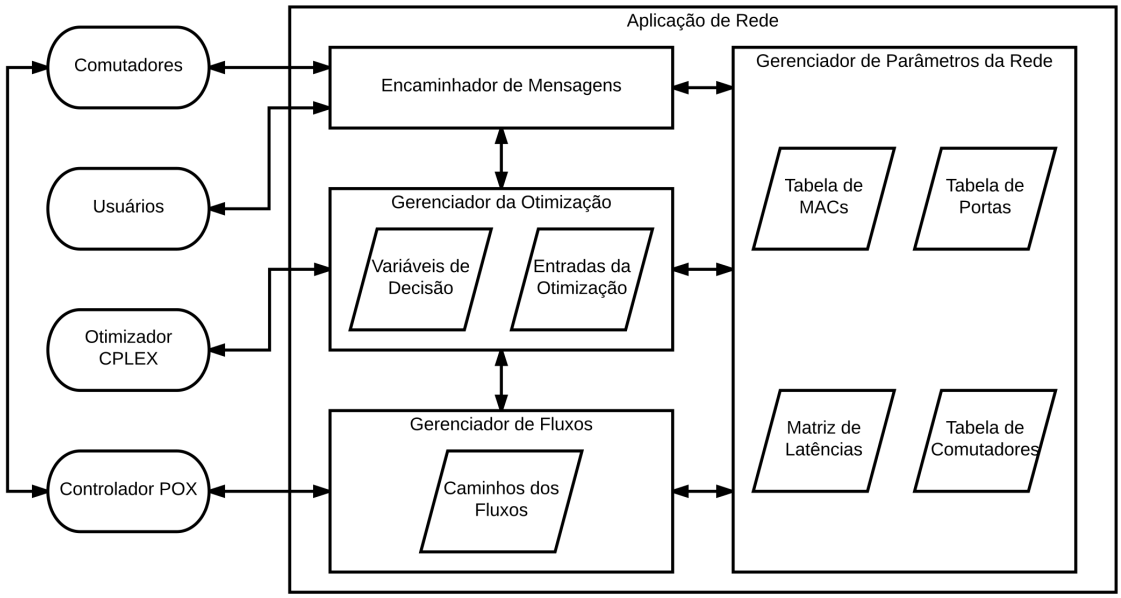


Figura 2. Diagrama de blocos da aplicação de rede.

O Encaminhador de Mensagens recebe as mensagens dos usuários e dos comutadores e também envia para os usuários mensagens com os valores de atrasos compensatórios. Além disso, por meio desse encaminhador, a aplicação de rede recebe os valores medidos de latência nos enlaces. Vale notar que, como detalhado mais adiante, esse encaminhador de mensagens pode não ser necessário. Isso é verdade visto que, em uma aplicação real de SDN, não é necessária a interação diretamente com o comutador, já que o controlador realiza esse papel. Além disso, os atrasos compensatórios podem ser inseridos em filas do comutador pelo próprio controlador, não necessitando de interação com os usuários.

O Gerenciador de Parâmetros da Rede possui estruturas de dados com as informações da rede necessárias para a aplicação. A Tabela de MACs possui o endereço MAC da estação de cada usuário. Apesar de neste trabalho serem utilizados MACs para identificar usuários, é possível modificar a aplicação para utilizar IPs. Na Tabela de Comutadores são armazenadas informações sobre os comutadores, indicando também quais usuários estão associados a cada comutador. Os comutadores são identificados por IDs de caminhos de dados openFlow (*OpenFlow Datapath ID* - DPIDs). O DPID é criado pelo POX e consiste em um identificador único do comutador [Dover, 2014]. A Tabela de Portas, por sua vez, contém informações das portas físicas dos comutadores utilizadas por cada enlace. Já a Matriz de Latências contém a latência de cada enlace e também informa se dois comutadores são vizinhos.

As Entradas da Otimização são as informações coletadas da rede e convertidas nas estruturas de dados aceitas pelo CPLEX. Essas informações são os conjuntos e parâmetros da Tabela 1, bem como os atrasos compensatórios inseridos em iterações anteriores. A cada iteração, o CPLEX calcula a solução do problema e a insere nas Variáveis de Decisão, que consistem nas variáveis da Tabela 1. Dessas variáveis, os atrasos compensatórios são enviados para o Encaminhador de Mensagens que, por sua vez, envia para os usuários a latência a ser inserida em cada um. Na implementação deste trabalho, os atrasos compensatórios são adicionados na própria estação do usuário. Isso é realizado para simplificar o desenvolvimento da aplicação. Entretanto, em um cenário real, esses atrasos devem ser inseridos em filas dos comutadores. As demais saídas do CPLEX são enviadas para o Gerenciamento de Fluxos, que grava os caminhos na estrutura de dados Caminhos dos Fluxos. O Gerenciador de Fluxos é então responsável por interagir com o POX para configurar esses caminhos na rede. Outra função desse bloco é configurar a rede para obter os parâmetros de rede iniciais, isto é, antes de o CPLEX calcular os caminhos pela primeira vez.

4.2. Funcionamento da aplicação de rede

A Figura 3 apresenta o diagrama de estados da aplicação de rede. No estado de descoberta, a aplicação de rede coleta informações que são constantes, como as que são inseridas na Tabela de MACs, Tabela de Portas e Tabela de Comutadores. Depois de coletados esses os dados, é realizado o reconhecimento da latência de cada enlace. Após isso, na etapa de otimização, são enviados os parâmetros para o otimizador, no qual as variáveis da Tabela 1 são calculadas. No estado de configuração recebe-se a solução do problema de otimização e configuram-se as tabelas de fluxo e os atrasos compensatórios. As três últimas etapas descritas na Figura 3 são realizadas periodicamente, de forma a verificar alterações nas latências dos enlaces como, por exemplo, quando uma carga maior

é enviada pela rede. Assim, os caminhos e atrasos compensatórios são reconfigurados de forma periódica. Os estados da aplicação são detalhados a seguir.

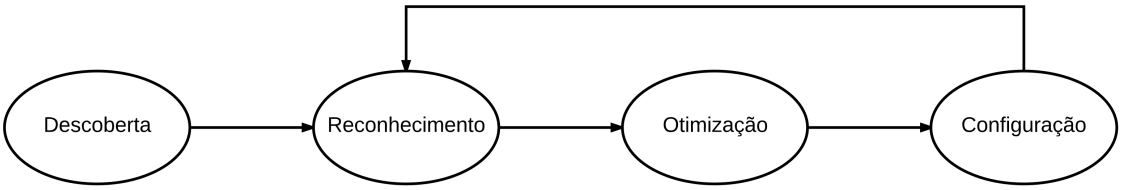


Figura 3. Diagrama de estados da aplicação de rede.

4.2.1. Descoberta

Os usuários são identificados à medida que se conectam ao servidor da rede. A identificação dos nós é realizada quando um *packet in* é detectado pelo Gerenciador de Fluxos. Um *packet in* é uma mensagem enviada por um comutador OpenFlow ao controlador. Caso o pacote seja o primeiro de um determinado endereço de origem detectado pelo Gerenciador de Fluxos, isso significa que o comutador que gerou o *packet in* é aquele ao qual o usuário ou o servidor está conectado. Assim, é possível associar um DPID a um usuário na Tabela de Comutadores. Durante todo o processo de descoberta, sempre que um *packet in* é analisado pelo Gerenciador de Fluxos, o número da porta física de entrada é armazenado na Tabela de Portas.

4.2.2. Reconhecimento

Esta etapa é responsável por calcular a Matriz de Latências. Para tal, enviam-se sondas entre os comutadores. Essas sondas são `pings` enviados a partir de estações (*hosts*) do Mininet. Assim, cada comutador possui uma estação de medição conectada a ele. A latência de um enlace é então definida como metade do tempo de ida e volta dos `pings` entre as estações dos comutadores conectados por esse enlace. Para esse cálculo, enviam-se três sondas de `ping` e realiza-se uma média dos valores de RTT obtidos. O uso dessas sondas também permite descobrir os vizinhos de um comutador, já que cada estação em um comutador envia `pings` para todos os comutadores da rede. Assim, esse considera como vizinho apenas os nós que responderam às sondas. A utilização de estações do Mininet para cálculo de latência é empregada neste trabalho apenas para facilitar o desenvolvimento da aplicação. Entretanto, em uma rede SDN real, a coleta das informações de latência deve ser realizada pelo próprio controlador, por meio uma aplicação de rede auxiliar.

4.2.3. Otimização

Nesta etapa, o Gerenciador da Otimização obtém os parâmetros da rede e os converte nas Entradas da Otimização. Essas entradas são então enviadas ao CPLEX. Esse otimizador, por sua vez, realiza os cálculos para solução do problema modelado na Seção 3.

Após a solução do Problema, enviam-se ao Gerenciador de Fluxos os caminhos de todos os usuários para o servidor e ao Encaminhador de Mensagens os atrasos compensatórios. Como apresentado anteriormente, esses atrasos são utilizados para compensar diferenças de latência entre os caminhos dos usuários. Entretanto, como o problema é executado de forma periódica e a latência dos enlaces varia na prática, a otimização tende a adicionar atrasos a cada iteração para compensar essa variação. Esse processo realizado sucessivamente, por várias iterações, leva a rede a um estado no qual as mensagens são perdidas devido aos grandes atrasos inseridos. Para evitar esse problema, subtrai-se do atraso compensatório atual o valor do atraso compensatório da iteração anterior. Dessa forma, o efeito aditivo de latência se reduz. Seja $\delta_k(n)$ o atraso compensatório calculado pelo otimizador na iteração n, o atraso $\Delta_k(n)$ a ser inserido efetivamente no usuário nessa iteração é dado pela Equação 5. Note que o atraso a ser inserido é zero se a diferença entre $\delta_k(n)$ e $\Delta_k(n-1)$ for negativa.

$$\Delta_k(n) = \begin{cases} \delta_k(n) - \Delta_k(n-1) & \text{se } \delta_k(n) > \Delta_k(n-1), \\ 0 & \text{caso contrário.} \end{cases} \tag{5}$$

4.2.4. Configuração

A saída do CPLEX indica quais enlaces devem ser configurados no caminho de cada usuário para o servidor. Assim, o Gerenciador de Fluxos recebe as informações do Gerenciador da Otimização e do Gerenciador de Parâmetros da Rede, define os fluxos e guarda todos os caminhos calculados na estrutura de dados Caminhos dos Fluxos. Quando um *packet in* é gerado para um determinado usuário, o POX interage com o Gerenciador de Fluxos para configurar as tabelas dos comutadores de acordo com os caminhos calculados. Em outras palavras, quando não há fluxo para um usuário em um comutador, o POX consulta os caminhos calculados. Assim, a configuração das tabelas dos comutadores é realizada de forma assíncrona a partir do envio de pacotes pelo usuário, independentemente da etapa (Figura 3) na qual a aplicação se encontra. Vale notar que um fluxo é definido em cada comutador pelo endereço MAC de origem do usuário.

Para garantir que os caminhos dos fluxos sejam alterados periodicamente com base nas decisões da aplicação de rede, toda entrada na tabela de fluxos é configurada com um *hard timeout*. Esse atributo indica por quanto tempo a regra deve permanecer instalada na tabela de fluxos. Quando esse tempo expira em um comutador para um determinado usuário, um *packet in* é enviado ao POX assim que uma mensagem desse usuário é recebida. Assim, o *hard timeout* é usado para forçar que os fluxos sejam apagados e, dado que o comutador precisa contatar o controlador, o problema de otimização é reexecutado.

O atraso compensatório, por sua vez, é informado a cada usuário por meio do Encaminhador de Mensagens. Diferentemente dos fluxos, esse atraso é configurado de forma síncrona, imediatamente após a etapa de Otimização. Na implementação, ao receber a mensagem com o atraso compensatório, a estação do usuário insere a latência por meio do comando TC do Linux. Obviamente, inserir o atraso no usuário pode se tornar uma falha de segurança, já que o mesmo pode desconfigurar as ações realizadas pelo comando TC, obtendo vantagens indevidas em termos de latência. Além disso, o

usuário pode não permitir o acesso à sua estação no nível de administrador. Entretanto, essa abordagem é utilizada neste trabalho apenas por simplicidade de implementação. No caso real, como já observado, esse atraso deve ser inserido no comutador de entrada do usuário por meio de filas de pacotes.

5. Avaliação de Desempenho

Para avaliar o desempenho da aplicação de rede proposta, foi utilizada a rede acadêmica da RNP [RNP, 2017]. A aplicação proposta foi comparada com o modo *Layer 2 learning* com *Spanning Tree* (L2-ST) já implementado no controlador POX. O modo L2 reproduz o comportamento de um comutador de nível 2 tradicional.

Para as emulações foi utilizado o Mininet; um sistema que permite emulação rápida de grandes redes em um único computador. Ele cria SDNs escaláveis usando mecanismos leves de virtualização. O Mininet possui recursos que criam, interagem, personalizam e compartilham os protótipos de rede criados rapidamente [De 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. Utilizou-se o Mininet para emular os dispositivos de repasse SDN e os usuários. A aplicação de rede proposta interage com o CPLEX, fornecendo os parâmetros da rede e aplicando os caminhos calculados. Nos experimentos deste trabalho, tanto a aplicação proposta como o modo L2-ST são executados pelo POX. As emulações e a execução do controlador foram realizadas em uma máquina virtual, usando o *software VMware Workstation 12* [VMware, 2012]. Essa máquina virtual possui sistema operacional Ubuntu, memória RAM de 4 GB e utiliza dois núcleos de um processador Intel Core i7-3770 com 3,40 GHz.

Neste trabalho a latência de cada usuário foi medida pelo RTT do usuário ao servidor. Para tal foram realizadas medições com o comando `ping`. Como apresentado na Seção 4, é necessário que haja múltiplas reconfigurações da rede ao longo do tempo, em função das modificações das condições no estado da rede devido ao tráfego. Assim, foram realizadas dez reconfigurações (iterações) em todos os casos em que o mecanismo proposto foi utilizado. Para os testes com o controlador no modo L2-ST, como a execução não é periódica, não há iterações. Cada teste foi realizado dez vezes, foi calculada a média dos RTTs médios dos usuários com um intervalo de confiança de 90%.

A topologia da rede da RNP e as bandas e latências dos enlaces utilizadas nas avaliações a seguir são apresentadas em [Couto et al., 2015]. Para escolher a posição de cada usuário na rede, 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 usuários. Logo, para o caso de dois usuários, um está conectado a um comutador posicionado em São Paulo e outro no Rio de Janeiro; para o caso de quatro usuários também são utilizados comutadores em Salvador e Fortaleza; por último, para oito usuários, são também usados comutadores em Belo Horizonte, Manaus, Curitiba e Recife.

Foram utilizadas no controlador cada uma das duas aplicações consideradas: L2-ST e com a justiça aplicada (proposta). Foram adicionados tráfegos TCP entre cada usuário e o servidor para simular tráfegos da aplicação para a qual a proposta procura diminuir as diferenças entre as latências dos usuários para o servidor. Para isso foi utili-

zada a ferramenta *Iperf*. Foram utilizados *Pings* para medir os RTTs.

A Figura 4 apresenta os RTTs de dois usuários (U1 e U2) em função do tempo para um dos testes do modo L2-ST. Na figura, é possível perceber uma grande variação do RTT. Além disso, os valores dos RTTs dos dois usuários para o servidor estão bem diferentes. Os picos de latência ocorrem devido a *hard timeouts*, ou seja, estão relacionados ao tempo em que uma regra fica instalada na tabela de fluxo do comutador (tempo de validade da regra). No caso de um *timeout*, há a necessidade de reconfiguração do comutador, ou seja, é necessário obter uma regra (que pode até ser a mesma anterior). Para isso, há a geração de *packet ins* e suas respectivas respostas, o que eleva o tempo de resposta (RTT).

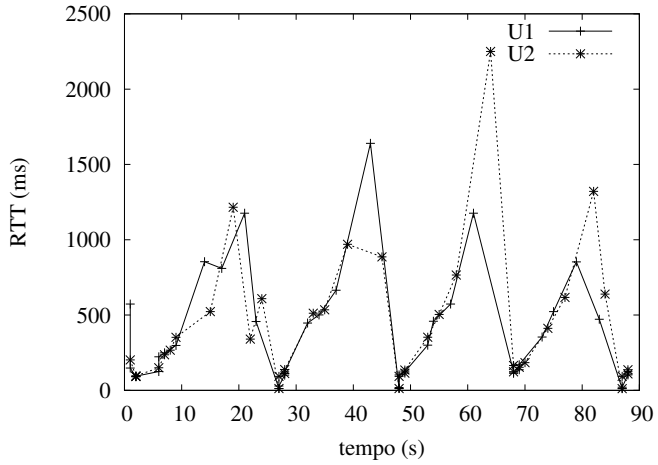


Figura 4. RTTs em função do tempo para o modo L2-ST – rede da RNP com dois usuários.

Na Figura 5, na qual o mecanismo proposto é utilizado, é possível observar que os RTTs dos usuários estão mais estáveis. 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 os fluxos são gerados. Percebe-se também que o RTT entre os dois usuários e o servidor no modo proposto estão mais próximos. Apesar de o problema de otimização forçar uma latência igual entre os usuários, na prática as latências são ligeiramente diferentes. Isso ocorre pois o modelo de otimização não considera aumento de latência causado por disputa de banda nos enlaces. Na mesma figura, os picos são causados principalmente pelos *timeouts* que disparam as reconfigurações. Consequentemente, seriam esperados mais picos, devido às sucessivas iterações. Entretanto, apenas dois picos são observados. Isso ocorre devido a uma possível falta de sincronismo entre o envio do *ping* de medição e a reconfiguração da rede.

As Figuras 6 e 7 apresentam os RTTs em função do tempo para um dos testes com quatro usuários. Como no caso anterior, os RTTs dos usuários para o servidor no modo L2-ST variam consideravelmente, enquanto que no mecanismo proposto os RTTs dos usuários se encontram bem próximos na maior parte do tempo.

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, tanto para oito quanto para dois e quatro usuários.

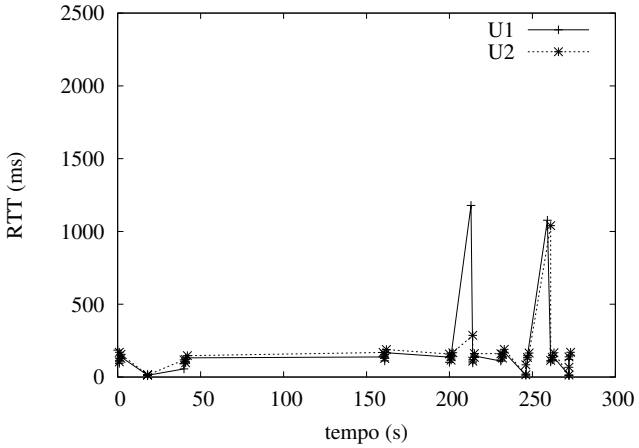


Figura 5. RTTs em função do tempo para modo justiça – rede da RNP com dois usuários.

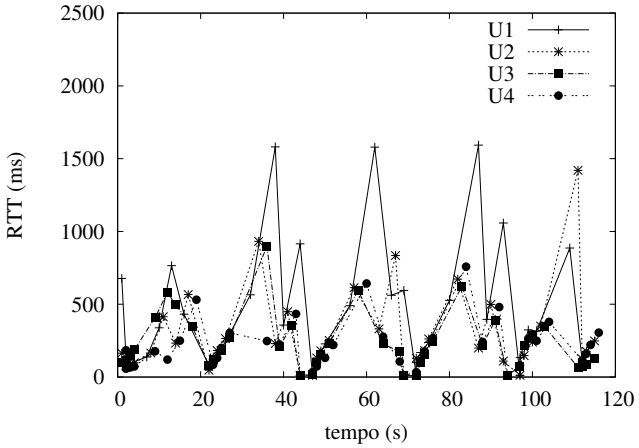


Figura 6. RTTs em função do tempo para o modo L2-ST – rede da RNP com quatro usuários.

As Figuras 8, 9 e 10 apresentam as médias dos RTTs médios de cada usuário com o controlador em ambos os modos. Para o caso com dois usuários (Figura 8), 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, já que duas cidades possuem valores similares de latência para o servidor. A redução da latência com o método proposto ocorre pois o cálculo de caminhos é realizado de forma a minimizar a latência. Essa redução, entretanto, seria facilmente obtida de forma tradicional, se o algoritmo Dijkstra fosse executado na rede.

Na Figura 9, a diferença entre os RTTs no modo L2-ST se dá devido à distância entre as cidades consideradas na análise. Já no modo de justiça é possível perceber que há uma menor diferença entre os RTTs de cada usuário para o servidor. Nota-se também que os RTTs de alguns usuários aumentam. Isso 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

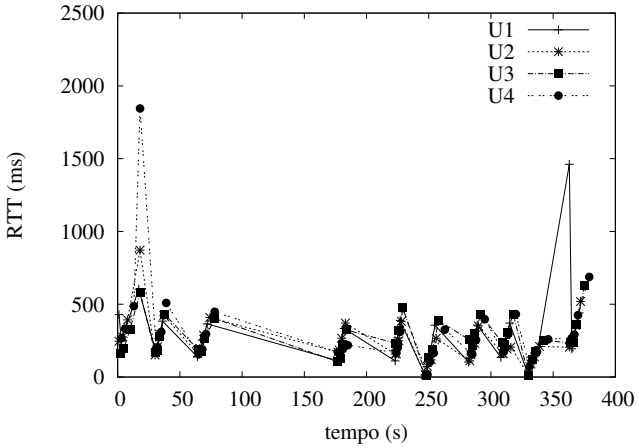
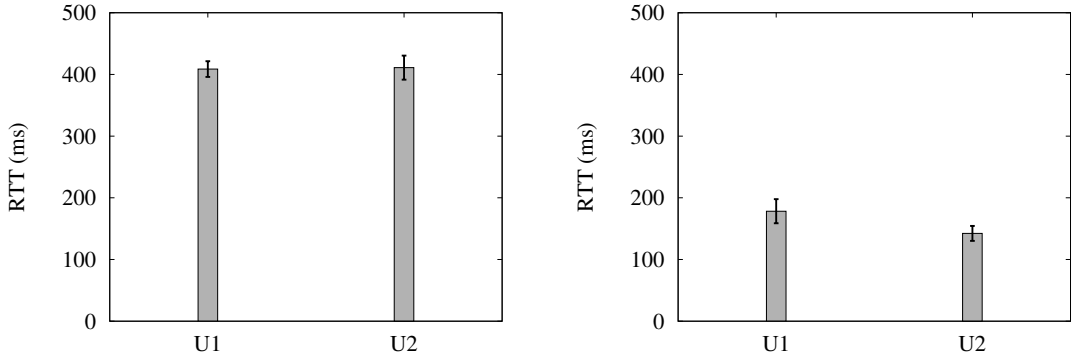


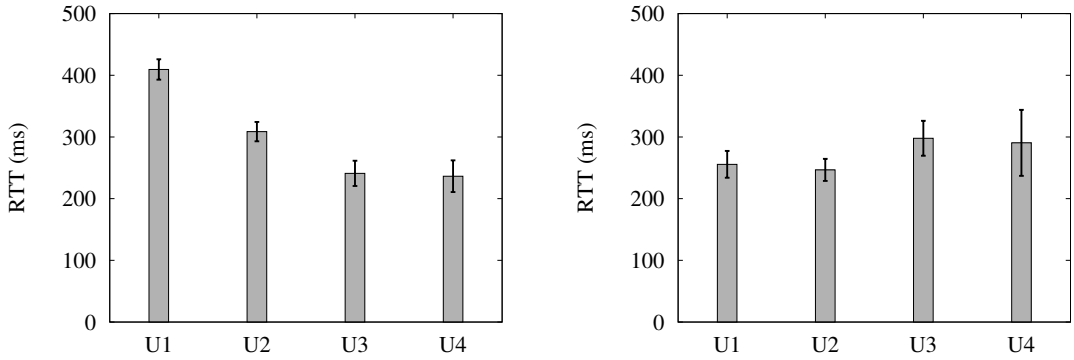
Figura 7. RTTs em função do tempo para modo justiça – rede da RNP com quatro usuários.



(a) L2-ST (b) justiça

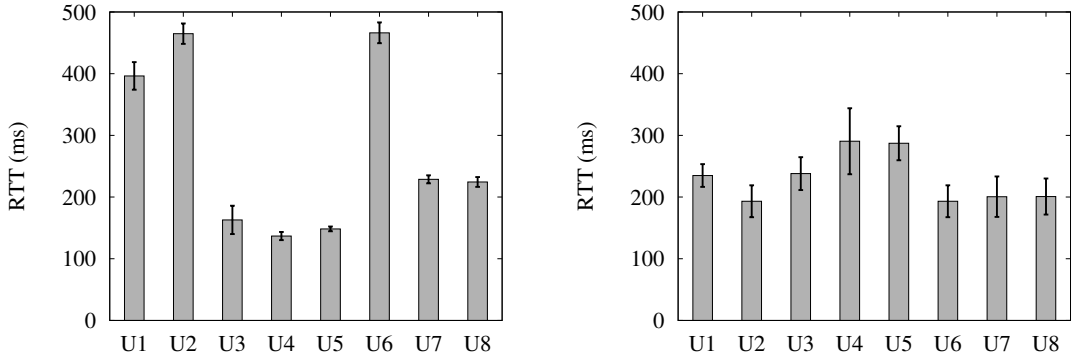
Figura 8. Média do RTT médio – rede da RNP com dois usuários.

Figuras 8 e 9, nota-se para o método proposto um aumento nos RTTs dos usuários U1 e U2 devido à inserção dos usuários U3 e U4. Na Figura 10, na qual o número de usuários é igual a oito, novamente o modo proposto consegue alcançar uma maior justiça .



(a) L2-ST (b) justiça

Figura 9. Média do RTT médio – rede da RNP com quatro usuários.



(a) L2-ST (b) justiça

Figura 10. Média do RTT médio – rede da RNP com oito usuários.

6. Conclusão

Este trabalho propôs uma aplicação para redes definidas por *software* que garante que as latências entre cada usuário e o servidor sejam as mais próximas possíveis. Essa aplicação consiste em resolver periodicamente um problema de otimização que leva em consideração a latência dos enlaces e calcula os caminhos dos fluxos na rede.

A partir dos resultados foi possível analisar o comportamento da latência dos usuários ao longo do tempo. Percebe-se que a aplicação proposta, quando comparada ao modo *layer 2 learning* com *spanning tree* do POX, consegue diminuir as diferenças entre as latências de cada usuário para o servidor. Pode-se notar nos testes realizados que, independentemente do número de usuários, é possível obter um cenário mais justo. Entretanto, o mecanismo proposto pode reduzir a satisfação de alguns usuários. Para evitar isso, mecanismos alternativos podem atuar em conjunto com o proposto, como impedir acesso a usuários com valores de latência muito altos ou dividir os usuários em grupos, de acordo com seus valores de latência, e aplicar a justiça separadamente em cada grupo.

De forma mais geral, este trabalho apresentou um estudo de caso de aplicação de uma rede SDN, que mostra que a centralização das decisões de encaminhamento fornece flexibilidade no desenvolvimento de esquemas de engenharia de tráfego. Em contrapartida, utilizar um otimizador como o CPLEX em redes tradicionais pode não ser trivial.

Nos resultados obtidos foi possível observar que as latências dos usuários com a aplicação proposta são próximas mas não iguais, como idealizado na modelagem da otimização. Isso ocorre pois o modelo não considera a disputa por banda nos enlaces. Assim, como trabalhos futuros, pretende-se considerar a banda dos enlaces no problema de otimização. Outra direção é propor uma heurística para realizar a etapa de otimização, visto que a obtenção da solução ótima pode não escalar para redes grandes.

Referências

Achterberg, T. e Berthold, T. (2007). Improving the feasibility pump. *Discrete Optimization*, 4(1):77–86.

Brun, J., Safaei, F. e Boustead, P. (2006). Managing latency and fairness in networked games. *Communications of the ACM*, 49(11):46–51.

Couto, R. S., Secci, S., Campista, M. E. M. e Costa, L. H. M. K. (2015). Otimização do posicionamento de servidores físicos em centros de dados resilientes a desastres. Em *XXXIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, p. 417–430.

De Oliveira, R. L. S., Shinoda, A. A., Schweitzer, C. M. e Prete, L. R. (2014). Using mininet for emulation and prototyping software-defined networks. Em *IEEE Colombian Conference on Communications and Computing (COLCOM)*, p. 1–6.

Debroy, S., Ahmad, M. Z., Iyengar, M. e Chatterjee, M. (2013). Critical sections in networked games. Em *IEEE International Conference on Communications (ICC)*, p. 2365–2369.

Dover, J. M. (2014). A switch table vulnerability in the open floodlight SDN controller. Relatório técnico.

Foundation, L. (2017). OpenDaylight. https://www.opendaylight.org/ - Acessado em dezembro de 2017.

Gorlatch, S. e Humernbrum, T. (2015). Enabling high-level QoS metrics for interactive online applications using SDN. Em *International Conference on Computing, Networking and Communications (ICNC)*, p. 707–711.

Huang, S. e Griffioen, J. (2013). Hypernet games: Leveraging SDN networks to improve multiplayer online games. Em *18th International Conference on Computer Games: AI, Animation, Mobile, Interactive Multimedia, Educational & Serious Games (CGAMES)*, p. 74–78.

Humernbrum, T., Glinka, F. e Gorlatch, S. (2014). A northbound API for QoS management in real-time interactive applications on software-defined networks. *Journal of Communications*, 9(8):607–615.

Li, W., Qi, H., Li, K., Stojmenovic, I. e Lan, J. (2017). Joint optimization of bandwidth for provider and delay for user in software defined data centers. *IEEE Transactions on Cloud Computing*, 5(2):331–343.

Maxemchuk, N. F., Shur, D. H. et al. (2001). An internet multicast system for the stock market. *ACM transactions on computer systems*, 19(3):384–412.

Mininet (2017). An instant virtual network on your laptop (or other pc). http://mininet.org/ - Acessado em dezembro de 2017.

POX (2017). Pox repository. https://github.com/noxrepo/pox - Acessado em dezembro de 2017.

RNP (2017). Rede ipê. https://www.rnp.br/servicos/conectividade/rede-ipe - Acessado em dezembro de 2017.

VMware (2012). Vmware - official site. https://www.vmware.com/ - Acessado em dezembro de 2017.