Comparação de Políticas de Divisão de Tráfego em Data Center empregando SDN

Erik de Britto e Silva[1,2], Henrique Moura[1], Daniel Fernandes Macedo[1], Luiz F. M. Vieira[1], Marcos A. M. Vieira[1]

[1]Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte, MG – Brasil

{erik,henriquemoura,damacedo,lfvieira,mmvieira}@dcc.ufmg.br

[2]Departamento de Computação e Sistemas
Universidade Federal de Ouro Preto (UFOP) – Campus João Monlevade
João Monlevade, MG – Brasil

erik@decsi.ufop.br

Abstract. *It is estimated that Internet traffic will triple in five years, which will increase server response time. One way to reduce such time is to balance the load on replicated servers. This work compares five load balancing policies using Software Defined Networks (SDN) in an OpenFlow switch: round robin, random, txbytes (transmitted bytes), cpuq-load (CPU usage and number of open connections), and load/load-prev (load forecast with switch statistics). These policies consider limitations such as the cost to retrieve network statistics and to install new rules. The results show that the txbytes and cpuq-load policies outperformed the others. On the other hand, the load-prev policy proved to be promising when adjusted for traffic.*

Resumo. *Estima-se que o tráfego da Internet triplicará em cinco anos, o que aumentará o tempo de resposta dos servidores. Uma forma de reduzir esse tempo é balancear a carga em servidores réplica. Este trabalho compara cinco políticas de balanceamento de carga em Redes Definidas por Software (SDN) em um switch OpenFlow: round robin, random, txbytes (bytes transmitidos), cpuq-load (taxa de CPU e número de conexões abertas) e load/load-prev (previsão com estatísticas do switch). As políticas consideram limitações como o custo para obter estatísticas da rede e instalar novas regras. Os resultados mostram que as políticas txbytes e cpuq-load superaram as demais. Já a política load-prev mostrou-se promissora ao se fazer ajustes em função do tráfego.*

1. Introdução

O volume global de dados trafegados na Internet tem crescido, como mostra o relatório *Cisco Visual Networking Index*[1]. O relatório prevê que o tráfego ultrapassará 1,1 Zettabytes (10^{21} bytes) em 2016, alcançando 2 Zettabytes em 2019. O tráfego global na nuvem (*cloud computing*) se multiplicará cerca de 3 vezes entre 2014 e 2019, passando de 3,4 Zettabytes para 10,4 Zettabytes [2]. Portanto, o tráfego dos data centers é o de maior

[1]http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html

[2]http://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/Cloud_Index_White_Paper.pdf

magnitude entre os citados.

Segundo [Cardellini et al. 2002], o desempenho das aplicações WEB percebido pelos usuários finais está sendo dominado pela latência dos servidores e, ao mesmo tempo, a capacidade da rede está aumentando mais rapidamente do que capacidade dos servidores. Isto indica que o uso combinado de vários servidores respondendo pelo mesmo serviço em um enlace de alta velocidade é adequado para se obter uma solução. Os desafios são, portanto, atender o volume futuro de tráfego nos data centers, bem como o crescente tráfego atual e suas oscilações, através do aumento da velocidade. Apenas aumentar a capacidade dos enlaces físicos para aumentar a taxa de transmissão dos dados, é de alto custo, não é escalável e implica no emprego de servidores mais rápidos, portanto mais caros. Uma alternativa é utilizar vários enlaces paralelos entre origem e destino, cada enlace conectado a um servidor réplica e empregar um método para a divisão do tráfego total entre os enlaces. Estes servidores réplica podem ser servidores de prateleira sem grandes modificações [Al-Fares et al. 2008], portanto de menor valor do que servidores especiais com características de maior desempenho.

Mesmo com mecanismos de otimização do atendimento ao tráfego, data centers são provisionados com capacidade em excesso [Xu et al. 2014] e sem escalabilidade. Fora do horário de pico ocorre o problema da subutilização de recursos, o que causa desperdício [Armbrust et al. 2010]. Empresas menores podem obter soluções escaláveis sem desperdício, como por exemplo o *AWS — Amazon Web Services* da Amazon [Ferraris et al. 2012], que oferece serviços e infraestrutura completa de data center por demanda. Porém, tais soluções podem não atender a todos os perfis de instituições. Portanto, algumas soluções são implementadas através de investimentos em infraestrutura física própria, que pode utilizar as políticas implementadas neste trabalho.

Em redes definidas por software (SDN) [Guedes et al. 2012, Macedo et al. 2015], o plano de controle (onde são tomadas as decisões de encaminhamento) é separado do plano de dados (onde o encaminhamento é feito). O plano de controle é executado em um controlador que pode rodar algoritmos diversos e portanto pode ser utilizado para engenharia de tráfego. Para utilizarmos engenharia de tráfego em SDN, por exemplo, um controlador OpenFlow será o responsável pela seleção dos caminhos [Jain et al. 2013]. O uso do protocolo OpenFlow permite controlar diretamente os fluxos, possibilitando que parâmetros de gerência de tráfego (banda, caminhos, *QoS*, etc.) sejam utilizados. O custo típico de uma solução para divisão de tráfego em um *middlebox* de 20 Gbps está em torno de US$ 80.000,00 [Patel et al. 2013], o dobro do custo de um switch OpenFlow 40 Gbps com um controlador, tomando como base a solução de um switch OpenFlow Xtreme Networks X770 (US$ 38,400,00) e um controlador Dell Core™ i7-6700 com 16 GB de memória (US$ 709,00) - preços da AMAZON[3].

Este trabalho compara diferentes políticas de balanceamento de carga, as quais são baseadas em redes definidas por software e têm como principal motivação o crescente volume de tráfego de entrada em um data center. Tais políticas utilizam Engenharia de Tráfego [Leduc et al. 2006] e visam diminuir o tempo médio de duração das requisições de serviço, dividindo-as entre vários servidores réplica. Dessa forma, mais requisições podem ser atendidas em um mesmo intervalo de tempo. Para evitar a subutilização, po-

[3]http://www.amazon.com, 17/Dez/2016

derão existir implementações de mecanismos para desligamento de servidores e switches – economia de energia e de vida útil. Por ser ambiente SDN, os recursos físicos não utilizados podem ser alocados para outras aplicações ou serviços, virtualizados ou não.

Todas as políticas aqui apresentadas realizam monitoramento de carga, cuja definição varia de uma política para outra. A carga pode ser: (1) volume de dados já trafegados nas portas de um switch, (2) utilização de CPU e o número de conexões abertas em cada servidor, (3) volume de dados já trafegados na interface de rede de cada servidor, (4) volume de dados nas portas de um switch, predito por meio de estatísticas do switch. Para ilustrar os resultados foram escolhidas requisições de serviço HTTP para realizar os experimentos executados.

A contribuição do presente trabalho é a comparação de cinco políticas de balanceamento de carga através da utilização de Engenharia de Tráfego em Redes Definidas por Software em um switch OpenFlow comercial. As políticas consideram limitações como o custo para obter estatísticas da rede e instalar novas regras.

O restante deste artigo está organizado da seguinte forma: Na Seção 2 são apresentados os ambientes de trabalhos relacionados. A Seção 3 apresenta as soluções a serem comparadas. O ambiente experimental e os resultados obtidos são apresentados na Seção 4. Por fim, a Seção 5 conclui este trabalho.

2. Trabalhos Relacionados

De forma similar ao presente trabalho, Akyildiz *et al*. utilizam *Differentiated Services* (DIFFSERV) e MPLS (*Multiprotocol Label Switching*) para implementar *QoS* em um ambiente de testes com TCP/IP [Akyildiz et al. 2003]. Em MPLS, rotas de backup são exigidas, mesmo com Engenharia de Tráfego, para manter *QoS*. A vantagem da Engenharia de Tráfego com SDN é: (1) a agilidade de programar a rede para isolar os enlaces em falha; (2) adicionar escalabilidade e eficiência a *QoS*, pois cada pacote é encaminhado através de um fluxo; (3) não há o custo adicional de reservar banda para cada classe diferente de tráfego como em DIFFSERV; e (4) o desempenho é aumentado, pois, em DIFFSERV, se uma classe tem seu tráfego reduzido ou é removida, não há a liberação da banda para as outras classes de tráfego, em tempo real.

Existem outras soluções similares que utilizam SDN. O ambiente B4, apresentado por Jain *et al*., interliga data centers da Google distribuídos em alguns continentes empregando Engenharia de Tráfego em SDN [Jain et al. 2013]. O B4 visa manter uma alta utilização dos enlaces, priorizar tráfego e prover tolerância a perdas, reescalonando os fluxos perdidos. Neste trabalho a solução apresentada é avaliada em menor escala relativa a quantidade de equipamentos e a complexidade da rede. A solução proposta neste artigo não lida com tolerância a perdas, maiores atrasos ou realocação de banda. Contudo estas características podem ser incorporadas futuramente.

O DUET gerencia data centers que provêm serviços na nuvem, realizando o balanceamento do tráfego utilizando uma solução com switches SDN em software e outros switches não SDN [Gandhi et al. 2014]. O controlador DUET calcula as rotas e distribui a programação das rotas de ECMP (*Equal-Cost Multi-Path*) e de tunelamento para os switches físicos e os de software. DUET fornece dez vezes mais capacidade e dez vezes menos latência quando comparado com soluções inteiramente baseadas em switches de

software para data centers, seu custo é menor e a solução se adapta mais rápido à dinâmica da rede. Ao contrário das soluções apresentadas no presente trabalho, o DUET emprega redundância de switches para alta disponibilidade da rede.

O *"OpenFlow-based Server Load Balancing Gone Wild"* utiliza programação proativa, inserindo regras coringa, de todos os possíveis prefixos dos endereços IPs dos clientes, para os enlaces de um data center [Wang et al. 2011]. Assim, os possíveis fluxos já estão divididos entre os vários caminhos para os servidores réplica. Com isso, poucos pacotes são direcionados ao controlador, resultando em um mínimo impacto na vazão da rede. Tal programação proativa não ocorre nas soluções propostas no presente trabalho.

O *Plug-n-Serve* realiza balanceamento de tráfego dentro de uma rede local composta de switches Openflow implementados em ambiente real [Handigol et al. 2009]. O *Plug-n-Serve* acompanha a localização física e a carga de servidores réplica na rede local, os quais podem ser conectados ou desconectados em diferentes switches ao acaso. Ele também monitora o congestionamento da rede através de medições de latência. Tal solução difere do presente trabalho ao considerar o congestionamento da rede para monitorar a latência e encaminhar os pacotes pelos caminhos com menor tempo de resposta. A latência é um parâmetro que pode ser adicionado futuramente em nossas soluções, como peso em uma política de escolha do melhor caminho até um servidor réplica.

Rodrigues et al. apresentaram o balanceamento de carga SDN a partir da utilização de CPU em cada servidor réplica [Rodrigues et al. 2015]. A solução proposta pelos autores apresenta diversas características semelhantes à solução **cpuq-load** do presente trabalho, com duas características distintas: (1) Rodrigues et al. utiliza em seus experimentos um *Open vSwitch* [Pfaff et al. 2015] em ambiente virtual, ao passo que a solução **cpuq-load** utiliza dispositivos reais; e (2) o presente trabalho compara soluções que utilizam mais parâmetros de entrada para o balanceamento do tráfego, tais como o tráfego em cada porta do switch real, consumo da CPU e número de conexões no servidor réplica, e o número de bytes já transmitidos na interface de rede.

Tabela 1. Comparação de características de ambientes de trabalhos relacionados

	RE	TR	ESC	AD	AP TR	CONG
AMBIENTE DO PRESENTE TRABALHO	X	I	I	+	I	+
Engenharia de Tráfego em Rede Não SDN - DIFFSERV	I	X	X	+	X	X
Engenharia de Tráfego com SDN - B4	X	I	I	I	I	I
Balanceamento de Carga Híbrido com SDN - DUET	X	I	I	I	I	+
OpenFlow Gone Wild	X	I	I	+	X	+
Balanceamento SDN - Plug-n-Serv	X	I	I	+	I	I
Balanceamento de Carga Web em Redes Definidas por Software	X	I	I	+	I	+

LEGENDA = I: Implementada X: Inexistente +: Pode ser adicionada

A Tabela 1 compara as características do ambiente das soluções apresentadas neste trabalho, ambiente de Redes Definidas por Software com Engenharia da Tráfego, com os ambientes dos trabalhos relacionados. Atribui-se o conceito ***RE*** *– Reserva estática de banda* caso a banda disponível seja dividida em faixas estáticas de menores bandas. Já o conceito ***TR*** *– Tempo real* é atribuído se a resposta do sistema às variações é suficientemente rápido para o bom funcionamento do sistema. A solução é considerada ***ESC*** *– Escalável* se a capacidade do sistema varia para atender às flutuações do tráfego. A solução é de ***AD*** *– Alta disponibilidade*, quando o sistema tem mecanismo(s) de resiliência a

falhas. A característica de ***AP TR*** *– Atraso de programação em tempo real*, onde o atraso da aplicação é crítico e pode impactar o funcionamento do sistema. Por fim, solução é ***CONG*** *– Monitora o congestionamento*, se na divisão do tráfego, o congestionamento da rede é considerado. Embora o ambiente apresentado com as soluções do presente trabalho não possuam no momento **RE** e sim uma escalabilidade conforme a carga, esta característica, bem como **AD** e **CONG**, podem ser adicionadas em trabalhos futuros.

3. Soluções de Divisão de Tráfego

Nesta seção são descritas as soluções de divisão de tráfego do presente trabalho.

3.1. Arquitetura

Neste trabalho, o tráfego a ser dividido entre os servidores réplica origina-se em estações clientes que fazem requisições a um serviço em um endereço IP virtual, seguindo a estrutura mostrada na Figura 1. A solução proposta pode ser aplicada para qualquer serviço TCP ou UDP, contudo neste artigo focamos no serviço HTTP. O modo utilizado para encaminhar cada fluxo aos servidores réplica é similar a um sistema Web baseado em *cluster* [Cardellini et al. 2002], que encaminha os fluxos das requisições HTTP de cada cliente, reescrevendo o endereço IP virtual para o endereço IP físico interno de um servidor réplica através dos switches. O respectivo fluxo da resposta do servidor para o cliente também tem o seu endereço IP físico interno reescrito para o endereço IP virtual nos switches da rede ao ser encaminhado para o cliente. Os switches suportam o protocolo OpenFlow, permitindo ao controlador SDN executar uma política ativa de balanceamento de carga. Este controlador determina os caminhos e faz a alocação de fluxos aos mesmos.

A inserção de uma regra no switch OpenFlow ocorre quando surge o primeiro pacote de um novo fluxo de dados em uma porta do switch. Após calcular por quais enlaces o pacote trafegará, o controlador insere no switch a regra de ida deste primeiro pacote, que o encaminha para a porta calculada. Todas as requisições dos clientes são destinadas ao IP virtual que identifica um servidor HTTP virtual. Assim, a cada novo fluxo a política de engenharia de tráfego define o servidor réplica de destino e insere no switch uma regra que reescreve o endereço IP físico do servidor destino no lugar do endereço IP virtual para este fluxo. Para melhor entendimento, são exibidas abaixo duas sequências de fluxo com reescrita de endereço IP:

- Fluxo de tráfego com as requisições dos clientes
  Clientes → IP Virtual → Switch reescreve IP → IP do servidor réplica
- Fluxo de tráfego com as respostas dos servidores réplicas
  Servidor réplica → Switch reescreve IP para IP Virtual → Clientes

3.2. Políticas de Balanceamento de Carga

O balanceamento de carga pode ser executado como um sistema de controle aberto ou fechado. Utilizamos neste trabalho duas políticas clássicas de balanceamento que funcionam como um sistema aberto. Elas são **round robin** e **random**. A política **round robin** alterna sequencialmente os fluxos entre os servidores disponíveis. A política **random** divide aleatoriamente os fluxos entre os servidores disponíveis. Estas políticas, por serem muito simples, foram utilizadas como *baseline* para nosso trabalho. O balanceamento de carga pode ser modelado, também, como um sistema de controle fechado, que

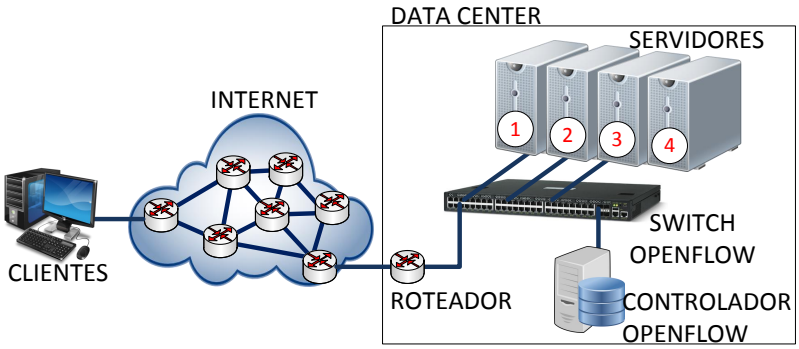


Figura 1. Solução com switch e controlador OpenFlow

utiliza uma variável de controle para atuar sobre o sistema. Neste trabalho apresentamos quatro políticas que utilizam controle de sistema fechado, que são descritas a seguir.

A política **txbytes** utiliza o volume de tráfego (bytes transmitidos) medido na interface da placa de rede de cada servidor. Ela seleciona o servidor com menor quantidade de bytes transmitidos, como mostrado no Algoritmo 1. Em caso de empate, é selecionado o servidor de menor IP. Para obter esta informação foi criada uma rede adicional e independente, conectando o controlador a cada servidor web utilizando um switch tradicional. Uma aplicação cliente-servidor em *sockets* foi implementada para obter os valores da interface do servidor desejado. O tráfego atual obtido sofre um atraso para ser lido, que dependerá da atividade do switch. Para diminuir este problema, pode ser utilizado um esquema de previsão de tráfego futuro, como em **load** e **load-prev-**δ.

Algoritmo 1 Algoritmo implementando a política **txbytes**

1: **function** TXBYTES($replicas$)
2: ▷ $replicas$: conjunto de servidores réplica físicos ordenados pelo IP
3: $load[i] \leftarrow sockcall\_get\_txbytes(replicas[i]), \forall i \in [1, |replicas|]$
4: **return** $\arg\min_i (load[i])$ ▷ servidor com menor carga

A política **cpuq-load** busca informações dos servidores da mesma maneira que **txbytes**, contudo são utilizados três parâmetros para seleção do servidor: a carga da CPU do servidor, o número de conexões na porta do serviço HTTP e o IP do servidor. O pseudocódigo é mostrado em Algoritmo 2. Os parâmetros são avaliados nesta ordem, isto é, selecionam-se os servidores com a menor carga de CPU, como mostrado na linha 4. $servers$ é uma lista dos servidores com carga de CPU mínima ordenada pelo IP. No caso da lista possuir mais de um elemento, é selecionado o servidor com menor quantidade de conexões e, por fim, o de menor número IP. Esta etapa é mostrada na linha 5. Assim, **cpuq-load** sofre o mesmo problema de atraso que **txbytes**.

A política **load** utiliza o volume de tráfego obtido pelo controlador mediante leitura regular das estatísticas do switch via primitiva *Portstats*. *Portstats* é uma chamada do tipo requisição-resposta assíncrona, definida pelo protocolo OpenFlow. O controlador envia uma mensagem solicitando informações sobre o tráfego nas portas do switch OpenFlow e este responde, posteriormente, ao controlador. A chegada da mensagem *Portstats* de resposta no controlador ativa uma função. Nossa aplicação executa a função apresentada em Algoritmo 3. A carga média, utilizada nas políticas **load** e **load-prev-**δ, é

Algoritmo 2 Algoritmo implementando a política **cpuq-load**

1: **function** CPUQ-LOAD($replicas$)
2: ▷ $replicas$: conjunto de servidores réplica físicos ordenados pelo IP
3: $load[i] \leftarrow sockcall\_get\_cpu\_connections(replicas[i]), \forall i \in [1, |replicas|]$
4: $servers \leftarrow \left\{ j \mid load[j].cpu = \min_{i=1}^{|replicas|} load[j].cpu \wedge j \in \{1, |replicas|\} \right\}$
5: **return** $\arg\min_i \ (load[i].num\_connections)$

calculada utilizando uma média móvel ponderada (*EWMA – Exponentially Weighted Moving Average*). A utilização desta média visa evitar a alta taxa de ocupação do controlador ao realizar várias leituras consecutivas em intervalos de tempos curtos, com o objetivo de manter o valor da carga instantânea, e também minimizar a disparidade dos valores utilizados como a carga no controlador entre duas leitura efetivas da carga real no switch [Guo et al. 2014]. Assim, a carga média atual é calculada na linha 10 do Algoritmo 3, onde $nova\_carga_i$ é o valor da carga obtida no instante i (linha 8), o valor da carga média anterior é $carga_{i-1}$ e $\alpha \in [0, 1]$ é o valor de ponderação. A *EWMA* utiliza, portanto, o peso α para ponderar amostras em ordem geometricamente decrescente. Neste trabalho, o valor da carga média é calculado a intervalos regulares de medição, quando a medição das estatístcas de *PortStats* (algoritmo 3) é executado. Portanto, durante o período entre o recebimento das estatísticas por *PortStats*, o valor da carga para cada servidor réplica registrado no controlador permanece constante. A cada novo fluxo, o controlador executa Algoritmo 4.

Algoritmo 3 Algoritmo de *PortStats*

1: **function** _HANDLE_PORTSTATS($\alpha, stats, t, replicas, cargas, tx$)
2: ▷ $stats$: estatísticas retornadas pelo switch ▷ α: fator de ponderação do EWMA
3: ▷ tx: lista com última leitura de bytes transmitidos ▷ t: período entre $PortStats$
4: ▷ $cargas$: carga média calculada anteriormente em $PortStats$
5: ▷ $replicas$: conjunto de servidores réplica físicos ordenados pelo IP
6: **for** $i \leftarrow 1, |replicas|$ **do**
7: $port \leftarrow replica[i].port$
8: $nova\_carga \leftarrow (stats.tx\_bytes[port] - tx[port])/t$
9: $tx[port] \leftarrow stats.tx\_bytes[port]$
10: $cargas[port] \leftarrow nova\_carga \times \alpha + cargas[port] \times (1 - \alpha)$

Algoritmo 4 Algoritmo implementando a política **load**

1: **function** LOAD($replicas, cargas$)
2: ▷ $replicas$: conjunto de servidores réplica físicos ordenados pelo IP
3: ▷ $cargas$: carga média calculada em $PortStats$ usando EWMA
4: **return** $\arg\min_i \ (cargas[i])$

A política **load-prev-δ** é similar a **load** ao utilizar *EWMA* para obter a carga média de cada servidor, porém, durante o período entre o recebimento das estatísticas via *PortStats*, acrescentamos uma previsão de fluxo à carga do servidor selecionado pelo controlador, como pode ser visto na linha 6 do Algoritmo 5. Assim, a cada recebimento de

Algoritmo 5 Algoritmo implementando a política **load-prev-**δ

1: **function** LOAD-PREV($replicas, cargas, \delta$)
2: ▷ $replicas$: conjunto de servidores réplica físicos ordenados pelo IP
3: ▷ $cargas$: carga média calculada em $PortStats$ usando EWMA
4: ▷ δ: acréscimo de carga
5: $j \leftarrow \arg\min_i (cargas[i])$
6: $cargas[replicas[j].port] += \delta$ ▷ Acrescenta previsão de carga ao servidor
7: **return** j

novo fluxo, quando o controlador determina qual servidor físico receberá o fluxo, o valor da carga média do servidor selecionado é incrementado por um valor δ, que representa o incremento de carga previsto a cada conexão. Nos experimentos realizados, o δ é fixo e definido na inicialização do controlador. Em trabalhos futuros pretendemos acrescentar um algoritmo de aprendizado para que o incremento seja alterado dinamicamente.

4. Avaliação Experimental

Essa seção foi dividida em duas partes: (1) Montagem do protótipo: descrição do hardware e software utilizado e a metodologia adotada na avaliação; e (2) Resultados e Análise: apresentação e análise dos resultados obtidos nos experimentos. A plataforma física simula um ambiente real similar ao mostrado na Figura 1. A seguir são descritos o hardware e software utilizados.

4.1. Montagem do protótipo

O protótipo consiste de uma estação que gera a carga dos clientes, um switch OpenFlow e quatro servidores. A estação é um computador Intel Core i5-2310 2,9 GHz, 8 GB RAM DDR3 e placa de rede Gigabit Ethernet. Os servidores possuem CPU Intel Core i7-3770 3,4 GHz, 16 GB RAM DDR3 e placa de rede Gigabit Ethernet rodando Apache. Nos experimentos foram utilizados quatro servidores web físicos. O Controlador é um Sony Vaio VGN-C240E com CPU Intel Core2 Duo T5500 1,66 GHz, 2 GB RAM DDR2 e uma placa de rede Ethernet 10/100 Mbps. O switch é um Pica8 P-3297, firmware 2.73, com 24 portas Gigabit Ethernet e OpenFlow 1.0. O Pica8 foi escolhido por ser um switch SDN de alto desempenho, que é empregado em diversas plataformas experimentais de SDN, tais como o FIBRE[4] e o FUTEBOL[5].

O controlador implementa os algoritmos de controle, que executam sobre o POX 0.3.0 (Carp). Na inicialização, os algoritmos limpam a tabela de fluxos do switch. Novos fluxos HTTP são tratados pelo algoritmo de balanceamento de carga, enquanto fluxos de outros tipos são tratados utilizando um switch l2_learning do POX.

As estatísticas de *PortStats* são amostradas em intervalos fixo de $10\ ms$ entre cada disparo. Valores inferiores a $5\ ms$ geram uma sobrecarga no switch Pica8, provocando a desconexão do canal de comunicação de controle entre o controlador e o switch. Para as políticas **txbytes** e **cpuq-load**, a aplicação em *sockets* é executada a cada *PacketIn*, ou seja, o controlador busca a informação de todos os servidores web a cada novo fluxo.

[4]http://www.fibre.org.br
[5]http://www.ict-futebol.org.br

No cliente são registradas quantas requisições foram solicitadas, o tempo médio de cada requisição (tempo de conexão), o tempo total, erros, conexões perdidas e a vazão média. O objetivo é usar o tempo médio de resposta para cada requisição para avaliar as políticas de Engenharia de Tráfego.

Os experimentos utilizaram 20, 60, 80, 100, 120, 160, 180 e 200 conexões simultâneas ao endereço do servidor virtual. Os valores de 140 e 160 conexões simultâneas não foram avaliados por estarem dentro de um intervalo de respostas lineares do switch, conforme pode ser observado entre 100 e 160 conexões em todas as curvas exibidas. Não foi utilizado um maior número de conexões simultâneas porque o switch utilizado apresentou um princípio de queda no desempenho com mais conexões simultâneas. Esses resultados corroboram com os resultados de desempenho de outros switches na literatura [Costa et al. 2016]. Foram realizados 33 experimentos para cada configuração, e os resultados foram avaliados para um intervalo de confiança de 95%. O valor de δ foi ajustado empiricamente para a carga com 20 conexões simultâneas. Verificamos que neste faixa, o melhor valor é $\delta = 13$ KB/s. Este valor é bastante inferior à taxa de transferência da rede que ficou em média 1080 KB/s com 20 conexões simultâneas.

4.2. Carga homogênea

Neste cenário utilizamos o HTTPERF[6] para simular o tráfego de vários clientes. A carga é homogênea, pois cada requisição solicita um arquivo de 100 KB. O tempo de resposta apresentado nas figuras 2, 3, 4 e 5 corresponde ao tempo total medido entre a requisição da URI feita pelo cliente e o recebimento completo do arquivo correspondente.

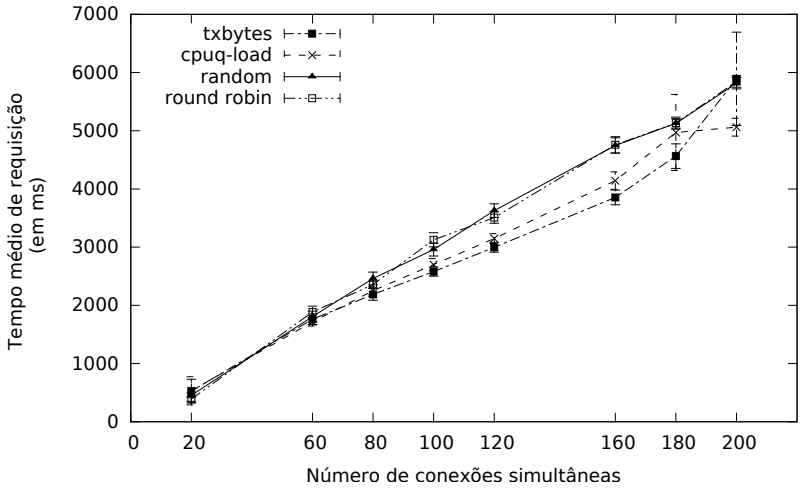


Figura 2. Comparação do *baseline* com políticas que buscam dados via *sockets*.

A Figura 2 mostra uma comparação do tempo médio de resposta utilizando as políticas clássicas de **round robin** e **random** com as políticas **txbytes** e **cpuq-load**, as quais buscam informações diretamente do servidor. Observa-se que com poucas conexões as quatro políticas são estatisticamente iguais. Contudo, à medida em que o número de conexões aumenta, a política **txbytes** sofre um aumento no tempo para tomada de decisão pelo controlador. Apesar disso, há uma melhora no desempenho percebido pelos clientes,

[6]https://github.com/httperf/httperf

o que é refletido na diminuição do tempo de resposta. Essa mesma melhora, porém em menor intensidade, também pode ser percebida com o uso da política **cpuq-load**.

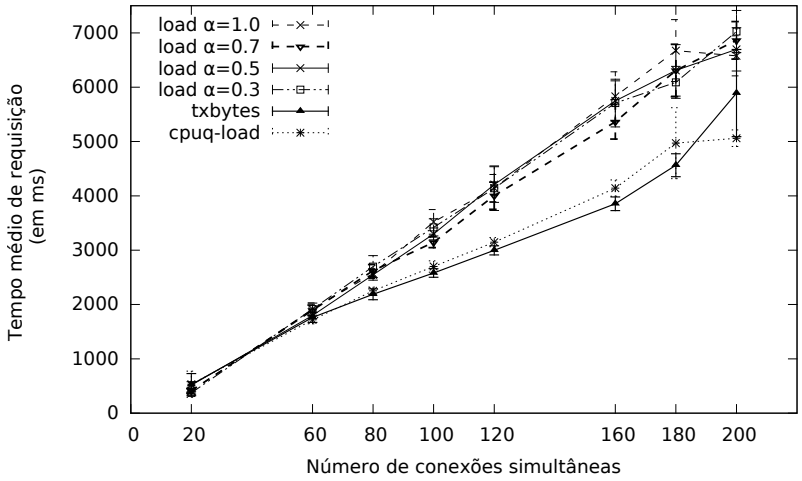


Figura 3. Comparação com política de carga via PortStats.

A política de previsão de carga **load** é comparada com **txbytes** e **cpuq-load** na Figura 3. As política **load** foi avaliada com valores de α variando em 0,3, 0,5, 0,7 e 1. Dessa forma verifica-se uma influência mais importante dos valores anteriores de carga para $\alpha = 0,3$ na carga média. Quando $\alpha = 1$, a carga média desconsidera completamente os resultados passados. Os melhores resultados entre as políticas **load** ocorreu para $\alpha = 0,7$. Isso confirma que a carga deve ser ajustada de forma a considerar mais fortemente o último tráfego medido, que corresponde às novas requisições que devem ser atendidas pelo servidor, entretanto, não é possível desconsiderar a carga medida no passado, pois o servidor web ainda pode estar processando parte dessa carga.

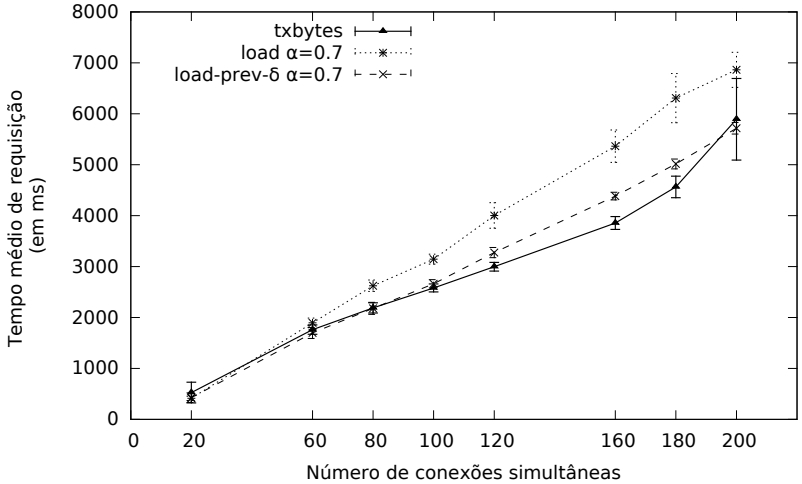
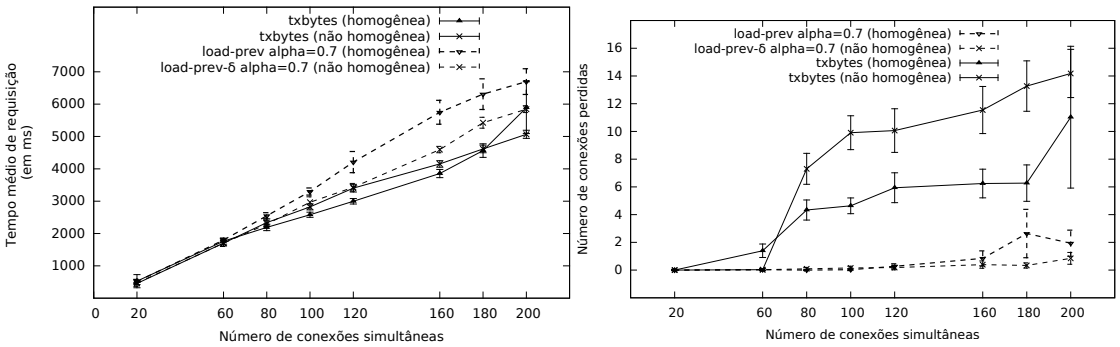


Figura 4. Comparação de políticas de carga com e sem previsão com txbytes.

Na Figura 4 é exibida uma comparação do desempenho da política **load** com **load-prev-**δ, ambas com $\alpha = 0,7$ que foi verificado anteriormente como o melhor valor de α para **load**. Estas duas políticas de previsão de carga são comparadas com **txbytes** que foi política que teve o melhor desempenho. Nota-se que com a previsão, os resultados

de **load-prev-δ** foram melhores que os demais na região próxima onde δ foi ajustado. A partir de 120 conexões, o valor **load-prev-δ** fica pior que **txbytes**, o que pode ser explicado por meio do ajuste do valor de δ para um número de conexões baixo. Para todos os casos avaliados, a política **load-prev-δ** é superior a **load**. Esse resultado era esperado pois **load** não considera a carga adicionada na rede durante o intervalo entre os *PortStats*, enquanto **load-prev-δ** considera o acréscimo de novo fluxo sobre a carga do servidor.



(a) Tempo médio de requisição. (b) Conexões perdidas

Figura 5. Comparação entre políticas com carga homogênea e heterogênea.

4.3. Carga heterogênea

Neste cenário simulamos uma carga heterogênea. Seguimos a metodologia de Summers *et al*. para geração de cargas de vídeo utilizando HTTPERF, baseado na utilização de sessões com *chunks* simulando o funcionamento de *Real-time Transport Protocol* (RTP) e *Real-Time Streaming Protocol* (RTSP) [Summers et al. 2012]. Foram configuradas sessões para o HTTPERF selecionando entre 588 blocos diferentes cujo tamanho varia até 2,3 vezes entre o menor e o maior bloco disponível. Foram criadas cerca de 100 sessões diferentes utilizando estes valores, desta forma são criadas sessões de tamanhos e sequências de requisições diferentes.

Na Figura 5(a) são exibidos os resultados obtidos por **txbytes** e **load-prev** com carga homogênea e com carga não homogênea. É possível observar que **txbytes** obtém os melhores resultados, indicando desta forma que a quantidade de bytes transmitidos é um melhor indicador para o desempenho. Isso se deve ao fato do desempenho do conjunto controlador e switch OpenFlow não ser capaz de atender adequadamente ao número de conexões. É possível verificar na Figura 5(b) que a perda média é zero para 20 conexões simultâneas, contudo, à medida em que o número de conexões simultâneas aumenta, o número de perdas de conexões também aumenta. É importante observar que o número de perdas de conexões para **txbytes** cresce mais rapidamente que em **load-prev**. Desta forma **txbytes** garante que as conexões que podem ser feitas sejam transmitidas pelo switch, obtendo um menor tempo de resposta, porém às custas de uma maior quantidade de perdas de conexão.

4.4. Discussão

O algoritmo que obteve melhor desempenho foi o **txbytes** que utiliza *sockets*. Embora os algoritmos **load** e **load-prev** obtiveram resultados próximos do **txbytes**, era esperado que eles tivessem melhor desempenho.

É esperado que a implementação da aquisição de estatíticas no switch OpenFlow fosse mais eficiente, e portanto mais rápida do que aplicações de sockets em cada servidor réplica. Porém, conforme os resultados pode-se concluir que as implementações físicas ainda devem evoluir para serem mais rápidas nos switches OpenFlow. O algoritmos que utilizam as estatísticas *PorStats*, mesmo não sendo disparados a cada PacketIn, mas em intervalos fixos de 10 *ms*, possuem um atraso devido às estas operações internas do switch. Com isto podemos supor que se a cada PacketIn as estatísticas fossem requisitadas ao switch, maiores atrasos seriam adicionados.

Pelas implementações e resultados obtidos pode-se confirmar a validade das soluções de divisão de tráfego neste trabalho. Novas gerações de dispositivos SDN e do protocolo OpenFlow podem ser esperadas para melhorar o desempenho. Sem dúvida, os switches SDN possuem requisitos que demandam maior desempenho do hardware conforme já publicado na literatura [Curtis et al. 2011], quando comparados com redes em arquiteturas tradicionais não SDN.

5. Conclusão e Trabalhos Futuros

Neste trabalho foram implementadas cinco políticas de divisão de tráfego utilizando uma rede OpenFlow, das quais duas são políticas clássicas – **round robin** e **random**, para fins de comparação e linha base (*baseline*). As políticas de previsão de carga baseadas na leitura de estatísticas do switch, via primitiva *PortStats* do OpenFlow, têm um pior desempenho do que a linha base devido ao atraso de execução desta primitiva. Embora haja uma melhora com a política de previsão load-prev-δ, o melhor valor para δ dependerá de características do hardware de cada switch e do perfil do tráfego, que não é conhecido previamente. As políticas cpuq-load e txbytes, que executam a leitura de carga diretamente nos servidores via sockets, são as que resultam em melhor desempenho. A política txbytes é melhor do que a cpuq-load por indicar valor mais significativo da carga e com os menores atrasos.

A implementação do protocolo OpenFlow nos switches requer maior rapidez computacional em várias ações, antes inexistentes nos switches tradicionais tal como se verifica, por exemplo, na leitura das estatísticas por fluxo e inserção de regras. As informações gravadas para estas ações ainda devem ser executadas num período de tempo menor do que o atingido pelo switch físico utilizado nos experimentos. Acreditamos que com o desenvolvimento de novas gerações de switches OpenFlow, será possível uma leitura das estatísticas em tempo suficiente para que possam ser utilizadas na política que é esperada ser a mais rápida dentre todas deste trabalho, a política load.

Como trabalhos futuros, pretendemos avaliar refinamentos às soluções apresentadas. Por exemplo, implementar políticas em que não utilizem reescrita de cabeçalho para reduzir o atraso, bem como empregar valores de δ variáveis. Além disso, novas plataformas SDN, tais como a PISA - *Protocol Independent Switch Architecture* - da empresa Barefoot Networks[7], serão avaliadas.

Referências

[Akyildiz et al. 2003] Akyildiz, I. F., Anjali, T., Chen, L., De Oliveira, J. C., Scoglio, C., Sciuto, A., Smith, J. A., and Uhl, G. (2003). A New Traffic Engineering Manager

[7] https://www.barefootnetworks.com/

for DiffServ/MPLS Networks: Design and Implementation on an IP QoS Testbed. *Comput. Commun.*, 26(4):388–403.

[Al-Fares et al. 2008] Al-Fares, M., Loukissas, A., and Vahdat, A. (2008). A Scalable, Commodity Data Center Network Architecture. *SIGCOMM Comput. Commun. Rev.*, 38(4):63–74.

[Armbrust et al. 2010] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. (2010). A View of Cloud Computing. *Commun. ACM*, 53(4):50–58.

[Cardellini et al. 2002] Cardellini, V., Casalicchio, E., Colajanni, M., and Yu, P. S. (2002). The state of the art in locally distributed web-server systems. *ACM Computing Surveys (CSUR)*, 34(2):263–311.

[Costa et al. 2016] Costa, L. C., Vieira, A. B., Silva, E. B., Macedo, D. F., Gomes, G., Correia, L. H., and Vieira, L. F. (2016). Avaliação de desempenho de plano de dados openflow. In *XXXIV Simpósio Brasileiro de Rede de Computadores, 2016*. SBC.

[Curtis et al. 2011] Curtis, A. R., Mogul, J. C., Tourrilhes, J., Yalagandula, P., Sharma, P., and Banerjee, S. (2011). Devoflow: Scaling flow management for high-performance networks. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, pages 254–265, New York, NY, USA. ACM.

[Ferraris et al. 2012] Ferraris, F., Franceschelli, D., Gioiosa, M., Lucia, D., Ardagna, D., Di Nitto, E., and Sharif, T. (2012). Evaluating the Auto Scaling Performance of Flexiscale and Amazon EC2 Clouds. In *International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 423–429.

[Gandhi et al. 2014] Gandhi, R., Liu, H. H., Hu, Y. C., Lu, G., Padhye, J., Yuan, L., and Zhang, M. (2014). Duet: Cloud Scale Load Balancing with Hardware and Software. In *ACM SIGCOMM*, pages 27–38.

[Guedes et al. 2012] Guedes, D., Vieira, L. F. M., Vieira, M. M., Rodrigues, H., and Nunes, R. V. (2012). Redes definidas por software: uma abordagem sistêmica para o desenvolvimento de pesquisas em redes de computadores. *Minicursos do Simpósio Brasileiro de Redes de Computadores*, 30(4):160–210.

[Guo et al. 2014] Guo, Z., Su, M., Xu, Y., Duan, Z., Wang, L., Hui, S., and Chao, H. J. (2014). Improving the performance of load balancing in software-defined networks through load variance-based synchronization. *Computer Networks*, 68:95–109.

[Handigol et al. 2009] Handigol, N., Seetharaman, S., Flajslik, M., and Johari, R. (2009). Plug-n-Serve: Load-balancing web traffic using OpenFlow. *Demo at ACM SIGCOMM*.

[Jain et al. 2013] Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M., Zolla, J., Hölzle, U., Stuart, S., and Vahdat, A. (2013). B4: Experience with a Globally-deployed Software Defined Wan. *SIGCOMM Comput. Commun. Rev.*, 43(4):3–14.

[Leduc et al. 2006] Leduc, G., Abrahamsson, H., Balon, S., Bessler, S., D'Arienzo, M., Delcourt, O., Domingo-Pascual, J., Cerav-Erbas, S., Gojmerac, I., Masip, X., Pescapè, A., Quoitin, B., Romano, S., Salvadori, E., Skivée, F., Tran, H., Uhlig, S., and Ümit,

H. (2006). An Open Source Traffic Engineering Toolbox. *Computer Communications*, 29(5):593 – 610.

[Macedo et al. 2015] Macedo, D. F., Guedes, D., Vieira, L. F. M., Vieira, M. A. M., and Nogueira, M. (2015). Programmable networks: From software-defined radio to software-defined networking. *IEEE Communications Surveys Tutorials*, 17(2):1102–1125.

[Patel et al. 2013] Patel, P., Bansal, D., Yuan, L., Murthy, A., Greenberg, A., Maltz, D. A., Kern, R., Kumar, H., Zikos, M., Wu, H., et al. (2013). Ananta: cloud scale load balancing. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 207–218.

[Pfaff et al. 2015] Pfaff, B., Pettit, J., Koponen, T., Jackson, E., Zhou, A., Rajahalme, J., Gross, J., Wang, A., Stringer, J., Shelar, P., Amidon, K., and Casado, M. (2015). The design and implementation of Open vSwitch. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 117–130.

[Rodrigues et al. 2015] Rodrigues, C. P., Costa, L. C., Vieira, M. A. M., Vieira, L. F. M., Macedo, D. F., and Vieira, A. B. (2015). Avaliação de balanceamento de carga web em redes definidas por software. In *XXXIII Simpósio Brasileiro de Rede de Computadores, 2015*, pages 585–597. SBC.

[Summers et al. 2012] Summers, J., Brecht, T., Eager, D., and Wong, B. (2012). Methodologies for generating http streaming video workloads to evaluate web server performance. In *International Systems and Storage Conference (SYSTOR)*, pages 2:1–2:12.

[Wang et al. 2011] Wang, R., Butnariu, D., and Rexford, J. (2011). OpenFlow-based Server Load Balancing Gone Wild. In *USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, pages 12–12.

[Xu et al. 2014] Xu, D., Fan, B., and Liu, X. (2014). Efficient Server Provisioning and Offloading Policies for Internet Datacenters With Dynamic Load-Demand. *IEEE Transactions on Computers*, 99:1.