Balanceamento de Carga entre Caminhos utilizando Redes Definidas por Software

Alisson Cavalcante e Silva, Marcelo G. Rubinstein *

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

alisson2000rj@gmail.com.br,rubi@uerj.br

Resumo. *É comum que redes de datacenters utilizem o modelo de rede Ethernet com encaminhamento de pacotes por caminho único baseado no Spanning Tree Protocol (STP), de forma a evitar loops na rede. Porém, tal fato não permite a utilização de enlaces ociosos para diminuir o congestionamento e aumentar a largura de banda agregada da rede. Uma solução seria dividir a rede em sub-redes conectadas por roteadores. Porém, a utilização de roteadores aumentaria consideravelmente o custo final da rede [Costa et al. 2012]. Uma forma de contornar este problema é realizar o balanceamento de carga entre enlaces. Este trabalho apresenta uma proposta de mecanismo de balanceamento de carga entre caminhos utilizando redes definidas por software (SDNs - Software Defined Networks). O mecanismo proposto, denominado MLB (Multipath Load Balance), foi implementado em Python e submetido a uma avaliação de desempenho, na qual os resultados mostraram que possível aumentar em 95% o valor da largura de banda agregada e diminuir em cerca de 95,5% a perda de pacotes em comparação ao modo de funcionamento padrão do controlador de SDN OpenDaylight.*

Abstract. *It is common for datacenter networks to use the Ethernet network model with packet forwarding by a single path based on the Spanning Tree Protocol (STP), to avoid loops on the network. However, this fact does not allow the use of idle links to reduce congestion and increase the aggregate bandwidth of the network. One solution would be to divide the network into subnets connected by routers. However, the use of routers would considerably increase the final cost of the network [Costa et al. 2012]. One way to get around this problem is to perform load balancing between links. This work presents a proposal for a load balancing mechanism between paths using Software Defined Networks (SDNs). The proposed mechanism, named MLB (Multipath Load Balance), was implemented in Python and submitted to a performance evaluation, in which the results showed that it is possible to increase the aggregate bandwidth by 95% and decrease the packet loss by about 95.5% compared to the standard OpenDaylight SDN controller operating mode.*

1. Introdução

Os serviços virtuais ofertados na Internet pelas nuvens comerciais ou dentro das redes empresariais por nuvens corporativas estão sendo cada vez mais consumidos pelos usuários.

*Este trabalho foi realizado com apoio da Marinha do Brasil, da FAPERJ, do CNPq e da CAPES.

Hoje em dia, é comum aparecerem da noite para o dia novas aplicações hospedadas nos *datacenters* oferecendo serviços essenciais para o âmbito profissional e também de entretenimento. Tal crescimento tende a promover o aumento da estrutura de rede com a introdução de novos *hosts* e consequentemente de novos comutadores, indispensáveis para a expansão do domínio de rede. Em geral, com a expansão da rede são criados novos enlaces que geram redundância de caminhos. Desta maneira, ao dispor de enlaces redundantes é possível utilizá-los não somente durante as situações de indisponibilidade, mas como também no dia-a-dia realizando balanceamento de carga entre eles. Este tipo de cenário torna possível o aumento da vazão do tráfego de dados combinando a largura de banda de dois ou mais caminhos da rede.

Este trabalho propõe um mecanismo de balanceamento de carga entre caminhos, denominado MLB (*Multipath Load Balance*), utilizando SDN (*Software Defined Networks*). O fato de redes SDN apresentarem uma arquitetura centralizada permite que controlador tenha uma visão global da topologia de rede e com isso é possível computar todos os caminhos entre a origem e o destino. Outro ponto importante é que redes SDN são programáveis. Essa característica permite ao administrador de rede criar aplicações inteligentes, usando dados coletados da rede, que podem tomar decisões como, por exemplo, por qual caminho um fluxo de rede deve ser encaminhado. Além disso, as redes Ethernet, bastante empregadas em *datacenters*, utilizam o protocolo STP (*Spanning Tree Protocol*) para gerar uma árvore de cobertura com caminho único, que faz com que o tráfego de dados seja enviado de um ponto a outro na rede sem que ocorram *loops* gerados por caminhos redundantes [Singh et al. 2015]. Graças à sua visão global da rede, o controlador de SDN torna dispensável o uso do STP e assume a função de gerenciar os caminhos da rede.

O mecanismo MLB é baseado no mecanismo de balanceamento de carga proposto por Nayan Seth [Seth 2016, Hassan 2017]. Entretanto, o MLB difere do mecanismo de Seth por contar com uma função denominada "controle de comutação" que utiliza algumas premissas para realizar a troca de caminho, evitando trocas que levem a um ganho muito baixo. O controle de comutação verifica se a ocupação atual do caminho ultrapassa 50% da capacidade deste e se o potencial novo caminho computado apresenta uma ocupação pelo menos 10% menor do que a ocupação do caminho atual; em testes preliminares envolvendo outros valores, os resultados finais mostraram-se bastante semelhantes. Além disso, o MLB realiza a computação de caminhos com enlaces disjuntos, enquanto que o mecanismo de Seth usa enlaces não disjuntos.

Como forma de avaliar o desempenho do mecanismo proposto, este trabalho realiza uma avaliação de desempenho entre o mecanismo MLB, o mecanismo de Seth e o controlador de SDN OpenDaylight (ODL), este desprovido de balanceamento de carga. Os resultados obtidos mostram que o MLB apresentou melhores desempenhos na agregação da largura de banda e na perda de pacotes.

O trabalho encontra-se organizado da seguinte forma. A Seção 2 apresenta os trabalhos relacionados. Algumas características do roteamento multicaminhos são apresentadas na Seção 3. A Seção 4 descreve as implementações dos mecanismos de balanceamento, enquanto a Seção 5 apresenta a avaliação de desempenho e seus resultados. Finalmente, a Seção 6 conclui o trabalho.

2. Trabalhos Relacionados

Alguns trabalhos tratam de balanceamento de carga entre enlaces utilizando redes SDN.

Em [Bredel et al. 2014], os autores apresentam uma otimização de tráfego por multicaminhos, que faz uso do controlador Floodlight e se baseia em caminhos disjuntos. Utilizam um algoritmo de seleção de caminhos que escolhe o caminho com menor número de fluxos mapeados. O trabalho compara diferentes algoritmos de seleção de caminhos baseados em: *hash* dos campos do cabeçalho do pacote, escolha aleatória, *round-robin* ou quantidade de fluxos existentes no caminho. Os resultados apresentados mostram que uso do algoritmo baseado na quantidade de fluxos apresenta um melhor desempenho em relação aos demais. Entretanto, os autores deixaram de incluir no trabalho a seleção por caminho baseada na taxa de tráfego e no volume de tráfego, geralmente utilizados para avaliar o congestionamento nos enlaces.

Em [Ramdhani et al. 2016], os autores abordam a limitação do roteamento de caminho único imposto pelo uso do STP em redes Ethernet. Os autores apontam o uso do SDN como forma de ajudar no roteamento com multicaminhos. É proposto um balanceamento de carga com um controle de admissão de fluxo ativado sempre que a carga de dados no caminho atingir 80% da capacidade deste, o que ajuda a diminuir o congestionamento da rede. O modelo emprega estatísticas dos comutadores coletadas pelo controlador Ryu que ajudam na tomada de decisão. Uma avaliação de desempenho demonstra o bom desempenho do modelo proposto ao conseguir uma taxa de transferência mais estável e menor latência em relação ao roteamento de caminho único do STP. Contudo, o controle de admissão busca diminuir o congestionamento da rede aplicando uma política de descarte de pacotes.

[Mallik and Hegde 2014] apresentam uma proposta de balanceamento de carga dinâmico utilizando multicaminhos com um mecanismo de controle do congestionamento que calcula o nível de carga de dados nos caminhos para saber se os mesmos podem ser utilizados na seleção de caminhos. A aplicação em conjunto com o controlador Floodlight procura detectar se a carga do caminho está acima de um determinado limite e instrui os comutadores a encaminharem os fluxos por um caminho alternativo. Os autores apontam como diferencial do trabalho a capacidade do seu modelo em identificar e reagir imediatamente ao desequilíbrio de carga e ao congestionamento de tráfego. Os autores relatam que são computados todos os possíveis caminhos entre a origem e o destino, caracterizando o uso de caminhos com enlaces não disjuntos. Contudo, é conhecido que esse tipo de computação de caminhos apresenta características de baixa tolerância a falhas.

[Bhandarkar and Khan 2015] implementam um balanceamento de carga dinâmico que escolhe o caminho com maior largura de banda livre para rotear o tráfego. O trabalho apresenta uma avaliação de desempenho, com tráfego de dados gerado pela ferramenta Cbench, entre a solução implementada e o balanceamento de carga do controlador Floodlight baseado em *round-robin*. Segundo os autores, os resultados comprovam que com uso de sua solução em comparação com o balanceamento de carga *round-robin* foi possível manipular mais pacotes, como também diminuir a latência da rede. Contudo, o trabalho não apresenta o nível de congestionamento da rede, o que poderia ser feito mostrando a taxa de perda de pacotes do tráfego gerado.

Este trabalho, de forma similar às propostas de [Ramdhani et al. 2016,

Mallik and Hegde 2014], também propõe um mecanismo de balanceamento de carga baseado em SDN que conta com um controle de comutação entre enlaces ativado sempre que o nível de carga no caminho atingir um determinado limiar. Além disso, esse mecanismo ainda verifica também se o novo caminho possui maior largura de banda livre em relação a do caminho atual. Assim, de modo geral, ele evita que sejam realizadas comutações desnecessárias que poderiam levar ao congestionamento do novo caminho escolhido pelo balanceador de carga.

3. Roteamento Multicaminhos

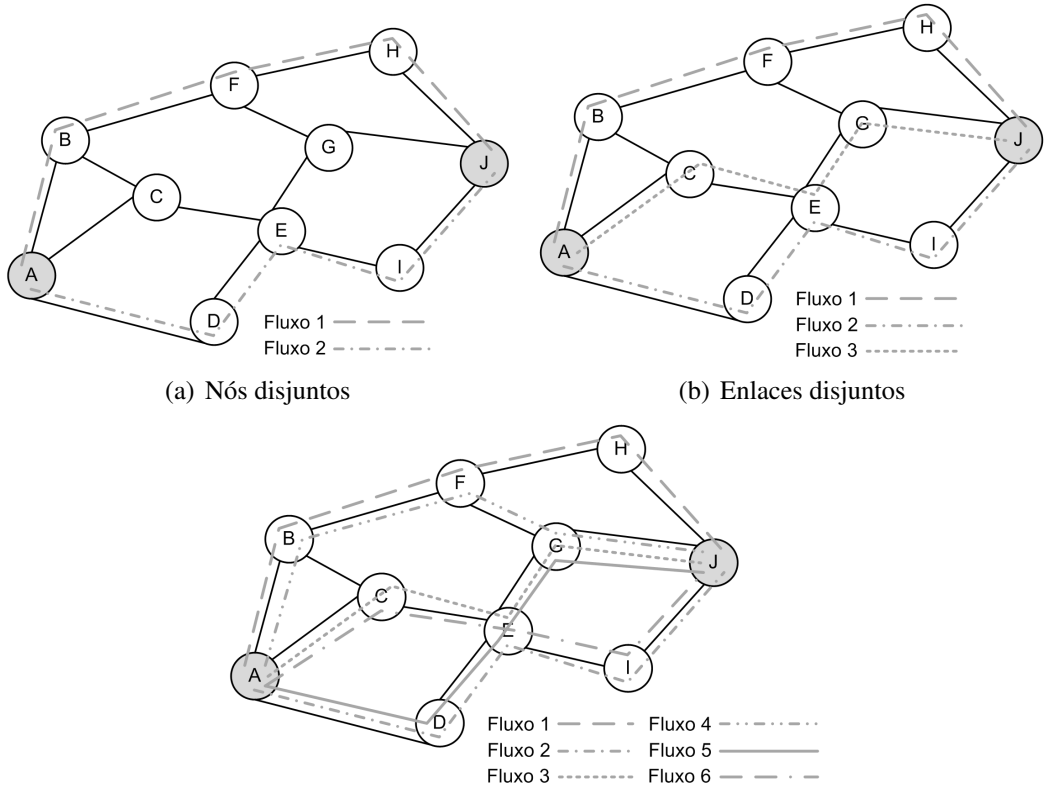
A técnica de roteamento por multicaminhos explora os recursos físicos da rede utilizando vários caminhos entre a origem e o destino para escoar o tráfego de dados [Tsai and Moors 2006]. A seguir são apresentados os três componentes básicos do roteamento multicaminhos: a computação de caminhos, a divisão de tráfego e a seleção de caminhos [Singh et al. 2015].

A computação de caminhos tem por objetivo encontrar todos os caminhos existentes entre uma origem e um destino. Para que o processo de busca seja eficaz, o algoritmo responsável pela computação de caminhos precisa ter conhecimento global da topologia da rede. Após a descoberta da topologia, o algoritmo precisa identificar os caminhos da origem até o destino com base em três cenários [Singh et al. 2015]:

a) nós disjuntos: o caminho é composto por nós (que não sejam a origem e o destino) não compartilhados por outros caminhos. O fato de não compartilhar nós significa que também não haverá compartilhamento de enlaces, exceto para enlaces *broadcast* e redes *Non-Broadcast Multi-Access* (NBMA). Esta configuração proporciona maior tolerância a falhas, já que os caminhos são totalmente independentes. Contudo implica maior gasto com infraestrutura, pois para se obter mais caminhos é necessário criar mais nós e enlaces. Esse cenário é exemplificado na Figura 1(a), que apresenta um grafo composto de 10 nós e 14 enlaces, sendo o nó "A" a origem e o nó "J" o destino de dois fluxos que percorrem caminhos totalmente independentes e livres do compartilhamento de nós e enlaces;
b) enlaces disjuntos: o caminho é composto por nós que podem ser compartilhados por outros caminhos, mas os enlaces não são compartilhados. Este cenário possui menor tolerância a falhas em relação ao cenário anterior. Isso porque, ao ocorrer falha em um nó, todos os caminhos que fizerem uso deste serão afetados também. O mesmo não acontecerá se a falha ocorrer em um enlace. O uso de enlaces disjuntos pode ajudar a aumentar a quantidade de largura de banda total e a diminuir o congestionamento na rede [Sun et al. 2012]. Um exemplo desse cenário é apresentado na Figura 1(b), onde que existem três fluxos percorrendo caminhos que não compartilham enlaces, mas apresentam compartilhamento do nó "E" pelos fluxos 2 e 3;
c) enlaces não disjuntos: exemplificado na Figura 1(c), tanto os nós quanto os enlaces de um caminho podem ser compartilhados pelos demais caminhos da rede. A falta de restrição quanto ao uso de nós e enlaces comuns torna mais fácil a computação dos caminhos [Singh et al. 2015]. Contudo este cenário é considerado o pior caso entre os três, pois caso ocorra uma falha em um nó ou enlace, essa falha afetará todos os caminhos que compartilharem tais recursos físicos.

O desempenho do algoritmo na computação de caminhos depende do número de nós e enlaces existentes na topologia. Segundo [Singh et al. 2015], estabelecer um número ideal de caminhos pode reduzir a complexidade do processamento empregado. Além disso, caminhos não disjuntos devem ser evitados, já que o compartilhamento de nós e enlaces significa ter que lidar com baixa tolerância a falhas. Outra questão é quanto à largura de banda, caminhos com enlaces compartilhados também terão a sua largura de banda compartilhada.

Os mecanismos de balanceamento de carga utilizados neste trabalho fazem uso de caminhos não disjuntos no caso do mecanismo de Seth e caminhos com enlaces disjuntos no caso do mecanismo proposto denominado MLB.



(a) Nós disjuntos

(b) Enlaces disjuntos

(c) Enlaces não disjuntos

Figura 1. Exemplos de cenários de computação de caminhos considerando o nó "A" como origem e "J" como destino.

Segundo [Prabhavat et al. 2012], a divisão de tráfego pode ser feita a nível de pacote, fluxo, subfluxo, superfluxo e sub-superfluxo. Neste trabalho utilizaremos a divisão de tráfego por fluxo baseada nos seguintes identificadores de pacotes: IP de origem, IP de destino, protocolo e porta de destino.

A seleção de caminhos pode ser classificada em função do tipo de seletor utilizado [Prabhavat et al. 2012]:

a) *round robin*: o tráfego é encaminhado por todos os caminhos computados obedecendo uma sequência cíclica;
b) informação do pacote: o tráfego é encaminhado com base em informações contidas no cabeçalho do pacote;

c) condição de tráfego: pode considerar a carga do tráfego, a vazão do tráfego, o volume do tráfego ou o número de fluxos ativos no caminho;
d) condição da rede: pode considerar o tamanho da fila, o atraso, o *jitter* ou a perda de pacotes no caminho.

As implementações de balanceamento de carga utilizadas neste trabalho se basearam na seleção de caminho por condição de tráfego, pois utilizam a vazão do tráfego no caminho (mecanismo MLB) e volume de tráfego no caminho [1] (mecanismo de Seth) como medidas de custo para identificar o melhor caminho.

4. Mecanismos de Balanceamento de Carga Avaliados

Neste trabalho foram utilizados dois mecanismos de balanceamento de carga entre caminhos, o proposto neste trabalho denominado MLB e o de Seth. Os dois mecanismos atuam na camada de aplicação e interagem com o controlador de rede utilizando sua *Northbounb Interface*. Além disso, utilizam a biblioteca "Networkx" escrita em python para criação, manipulação e estudo de grafos e redes [Hagberg et al. 2008]. A seguir será apresentado o funcionamento das implementações de ambos os mecanismos.

4.1. Implementação do Mecanismo de Seth

Esta implementação baseia-se no algoritmo de Dijkstra para computar múltiplos caminhos não disjuntos que apresentem o menor número de saltos entre a origem e o destino. A escolha pelo mecanismo de Seth deu-se pelo fato de seu código fonte estar disponível em diretório público sob licença de distribuição *General Public License* (GNU) para softwares livres [Seth 2016]. O pseudocódigo desta implementação é apresentado no Algoritmo 1.

Algoritmo 1: IMPLEMENTAÇÃO DE SETH

```
1  início
2      origem = host de origem
3      destino = host de destino
4      t = 60 segundos
5      enquanto (True) faça
6          nos_conectados = proc_topologia()
7          G = networkx.graph(nos_conectados)
8          caminhos = networkx.all_shortest(G, origem, destino, dijkstra)
9          para i de 1 até quantidade(caminhos) faça
10             t1 = retorna_dados_tx_rx(OpenDaylight_restconf, caminhos[i])
11             espera 2 s
12             t2 = retorna_dados_tx_rx(OpenDaylight_restconf, caminhos[i] )
13             custo_caminhos[i] = t2 - t1
14         fim
15         melhor_caminho = retorna_indice(min(custo_caminhos))
16         configura_fluxo(caminhos[melhor_caminho], origem, destino)
17         espera(t)
18     fim
19 fim
```

Os hosts de origem e destino são definidos no início da execução, como mostram as linhas 2 e 3. A implementação de Seth está estruturada de maneira a realizar balanceamento de carga de apenas um fluxo. Por isso, são definidos hosts de origem e destino.

[1] volume de dados acumulado no caminho durante transmissões.

Na linha 4 é definido o tempo de espera entre cada rodada de execução que será aplicado na linha 17. O tempo de 60 s foi definido pelo autor do algoritmo e mantido nos experimentos realizados. Na linha 5 tem início a estrutura de repetição condicionada à sentença "enquanto $(True)$ faça", que mantém a aplicação em constante execução sem definição de término. Na linha 6 a função "proc_topologia()" retorna a informação das conexões estabelecidas entre os nós. Já a linha 7 carrega a biblioteca "*NetworkX*" recebendo como parâmetro de entrada as informações da topologia armazenadas em "nos_conectados". Na linha 8, a estrutura de grafos "G" é processada pela função "all_shortest", baseada no algoritmo de Dijkstra. Como resultado, são retornados todos os caminhos computados entre origem e destino e armazenados na variável "caminhos". Na linha 9, tem início um *loop* que se repetirá dada a quantidade de caminhos computados. Dentro do laço, na linha 10, "t1" recebe do controlador, no tempo atual, as informações do volume de dados transmitidos e recebidos pelas interfaces dos comutadores que compõem o caminho e passados 2 s de espera, "t2" recebe também do controlador, no tempo atual, as mesmas informações do volume de dados, como mostra a linha 12. Então, na linha 13 o valor de "t2" é subtraído por "t1" e o resultado armazenado no vetor "custo_caminhos". A variável "melhor_caminho" recebe o caminho com menor volume de dados registrado, na linha 15. E finalmente, na linha 16, a aplicação submete ao controlador as informações do novo caminho que o fluxo deve utilizar.

4.2. Implementação do Mecanismo MLB

Esta implementação baseia-se no algoritmo de Edmonds e Karp [Cormen et al. 2012] que computa múltiplos caminhos com enlaces disjuntos que apresentem o menor número de saltos entre a origem e o destino. Ela se difere da implementação de Seth não somente pelo tipo de computação de caminhos, mas também pela quantidade fluxos que pode manipular. Além disso, o MLB possui um "controle de comutação" que verifica se a ocupação atual do caminho ultrapassa 50% da capacidade deste e se o potencial novo caminho computado apresenta uma ocupação pelo menos 10% menor do que a do caminho atual. O pseudocódigo desta implementação é apresentado no Algoritmo 2.

As linhas 2 e 3 definem os comutadores de origem e destino. Já a linha 4, define a quantidade de fluxos que serão balanceados. Nas linhas 5, 6 e 7 são definidos os valores de referência para uso pelo controle de comutação, que pode ser visto na linha 21 e será explicado adiante. Na linha 8 tem início a estrutura de repetição condicionada a sentença "enquanto $(True)$ faça", que mantém a aplicação em constante execução sem definição de término. Na linha 9 a função "proc_topologia()" retorna a informação das conexões estabelecidas entre os nós. Já a linha 10 carrega a biblioteca "*NetworkX*" recebendo como parâmetro de entrada as informações da topologia armazenadas em "nos_conectados". Já a linha 11 apresenta estrutura de repetição condicionada ao número de fluxos em uso; ou seja, todo processo é repetido para cada fluxo em uso.

Na linha 12, a estrutura de grafos "G" é processada pela função "edge_disjoint()" utilizando o método de computação de caminhos de Edmonds e Karp. Como resultado, são retornados os caminhos computados e armazenados na variável "caminhos". Na linha 13, tem início um *loop* que se repetirá dada a quantidade de caminhos computados. Neste, a função "calcula_ocupacao" estima a taxa de ocupação nos caminhos computados usando as informações de volume de dados transmitidos e recebidos pelos comutadores que compõem cada caminho, sendo armazenadas em "custo_caminhos", na linha 14. Na

linha 16, a variável "melhor_caminho" recebe o caminho com menor taxa de ocupação computada. E por fim, A linha 17, testa se é a primeira rodada de execução da aplicação para o fluxo corrente. Em caso positivo, o caminho para o fluxo é configurado na linha 18 e o vetor "atual" recebe o índice do melhor caminho para o fluxo corrente, conforme apresentado na linha 19. Em caso negativo, a linha 21 apresenta o controle de comutação que permite a comutação do fluxo para o novo caminho caso a ocupação do caminho atual seja maior ou igual ao valor definido na variável "condicao1", neste caso 50% da sua capacidade, e o novo caminho apresente uma ocupação menor em 10% ou mais da ocupação do caminho atual, valor definido na variável "condicao2". Para os casos em que as duas condições forem satisfeitas o novo caminho alternativo será configurado para o fluxo corrente na linha 22 e o vetor "atual" receberá o índice do melhor caminho para o fluxo corrente, conforme apresentado na linha 23.

Algoritmo 2: IMPLEMENTAÇÃO MLB

```
1  início
2      origem = comutador de origem
3      destino = comutador de destino
4      qtd_fluxo = quantidade de fluxo
5      condicao1 = 50%
6      condicao2 = 10%
7      capacidade_caminho = 10.000.000
8      enquanto (True) faça
9          nos_conectados = proc_topologia()
10         G = networkx.graph(nos_conectados)
11         para fluxo de 1 até qtd_fluxos faça
12             caminhos = networkx.edge_disjoint(G, origem, destino, edmonds_karp)
13             para i de 1 até quantidade(caminhos) faça
14                 custo_caminhos[i] = calcula_ocupacao(OpenDaylight_restconf,
                     caminhos[i])
15             fim
16             melhor_caminho = retorna_indice(min(custo_caminhos))
17             se atual[fluxo] = ∅ então
18                 configura_fluxo(caminhos[melhor_caminho], origem, destino, fluxo)
19                 atual[fluxo] = melhor_caminho
20             fim
21             senão se custo_caminhos[atual[fluxo]] maior em (condicao1) ou mais que
                   capacidade_caminho & custo_caminhos[melhor_caminho]
                   menor em (condicao2) ou mais que custo_caminhos[atual[fluxo]] então
22                 configura_fluxo(caminhos[melhor_caminho], origem, destino, fluxo)
23                 atual[fluxo] = melhor_caminho
24             fim
25         fim
26     fim
27 fim
```

5. Avaliação de Desempenho

Esta seção apresenta a avaliação de desempenho realizada utilizando o controlador ODL desprovido do uso de mecanismos de balanceamento de carga, com o uso do mecanismo de Seth e com o uso do mecanismo MLB. A seguir são apresentadas as métricas utilizadas na obtenção dos resultados, seguidas dos mecanismos utilizados e dos modelos de tráfego.

As métricas utilizadas foram:

a) vazão agregada: a utilização de balanceamento de carga tende a aumentar a largura de banda agregada. Essa métrica é calculada através do somatório das vazões dos fluxos observados no destino. Para o seu cálculo, foram utilizados fluxos compostos por pacotes TCP (*Transmission Control Protocol*);
b) justiça: o uso de um caminho por mais de um fluxo é considerado justo quando todos os fluxos fazem uso de partes iguais da largura de banda disponível. O índice de justiça retorna um valor compreendido entre 0 e 1. Se todos os fluxos tiverem uma taxa de vazão praticamente igual entre eles, o resultado se manterá próximo a 1. Se ocorrerem diferenças significativas nas taxas de vazão, o resultado tenderá a 0. O índice de justiça [Jain 1991] é dado por:

$$f(x_1, x_2, ..., x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} x_i^2}, \quad (1)$$

onde "n" é o número total de fluxos concorrendo pela largura de banda e "x_i" é o valor da vazão do fluxo "i";
c) perda de pacotes: a perda de pacotes é decorrente principalmente do congestionamento no caminho. A medição é feita no destino com base na diferença entre a quantidade de pacotes que foi enviada pela origem e o que foi recebido no destino. Neste caso foram utilizados fluxos compostos por pacotes UDP (*User Datagram Protocol*).

A seguir são apresentados com mais detalhes os três mecanismos utilizados no experimento:

a) ODL: único mecanismo desprovido de balanceamento de carga. A dinâmica de funcionamento deste mecanismo envolve apenas o uso do controlador ODL; ou seja, todos os fluxos são comutados pelo mesmo caminho padrão definido na árvore de cobertura;
b) de Seth: realiza balanceamento de carga utilizando a implementação de Seth baseada em caminhos não disjuntos;
c) MLB: realiza balanceamento de carga utilizando a implementação MLB baseada em caminhos com enlaces disjuntos.

O mecanismo de Seth realiza balanceamento de carga de apenas um dos fluxos, neste caso do fluxo principal. Já o mecanismo MLB pode realizar o balanceamento de mais fluxos. Contudo, para tornar justa esta avaliação, o MLB foi configurado também para balancear apenas o fluxo principal. Além disso, tanto no caso do ODL quanto nos casos dos mecanismos de balanceamento foi utilizado outro fluxo que corresponde a um tráfego de fundo. Dois tipos de modelos de tráfego foram utilizados:

a) Modelo TCP-2f: composto por dois fluxos, sendo o fluxo principal tendo como origem o host "h1" e como destino o host "s5" na porta TCP/5001 e o fluxo de fundo tendo como origem o host "h2" e "s6" como destino na porta TCP/5002 (ver Figura 2). A duração dos fluxos foi fixada em 600 s;
b) Modelo UDP-2f: composto por dois fluxos, sendo o fluxo principal tendo como origem o host "h1" e como destino o host "s5" na porta UDP/5001 e o fluxo de fundo tendo como origem "h2" e como destino "s6" na porta UDP/5002 (ver Figura 2). A duração dos fluxos também foi fixada em 600 s.

Os fluxos UDP foram configurados para transmitirem dados a uma taxa de 9,5 Mbps; ou seja, ocupando 95% da capacidade dos enlaces da rede. Essa taxa de transmissão foi escolhida após decorridos vários testes nos quais a perda de pacotes foi medida. A taxa de transmissão foi aumentada sucessivamente até que se chegasse ao limite de uma transmissão sem perda de pacotes; 9,5 Mbps nos experimentos. Todos os fluxos foram gerados utilizando a ferramenta iPerf. Cada teste foi executado 30 vezes. Os resultados foram processados com base no uso de média aritmética e intervalo de confiança de 95%.

A Figura 2 apresenta a topologia de rede composta por oito nós, três caminhos disjuntos e cinco caminhos não disjuntos que foi utilizada nos experimentos e emulada no Mininet [Lantz et al. 2010]. Os nós da rede foram emulados utilizando o comutador virtual Open vSwitch. A capacidade de todos os enlaces da rede foi configurada em 10 Mbps devido aos recursos computacionais disponíveis. O controlador OpenDaylight foi escolhido por se tratar de um projeto aberto já bastante disseminado no meio acadêmico e corporativo [Kreutz et al. 2015]. Desenvolvido na linguagem de programação Java, ele possui uma aplicação nativa denominada "l2switch" composta pelo módulo "*Loop Remover*" responsável por gerar a árvore de cobertura da rede, similar à gerada pelo STP. A árvore de cobertura trata-se de caminhos padrão que abrangem todos os nós da rede e visa eliminar caminhos redundantes que possam causar *loops* na rede. O ODL mantém um inventário da árvore de cobertura com o "*status*" do STP "*forwarding*" para os enlaces ativos e "*discarding*" para os enlaces inativos. No exemplo da figura, para o host "h1" se comunicar com o host "s5", é utilizado o caminho 1-2-6-8. Os *status* dos enlaces podem ser verificados por meio do console web de gerência do ODL [OpenDaylight 2017].

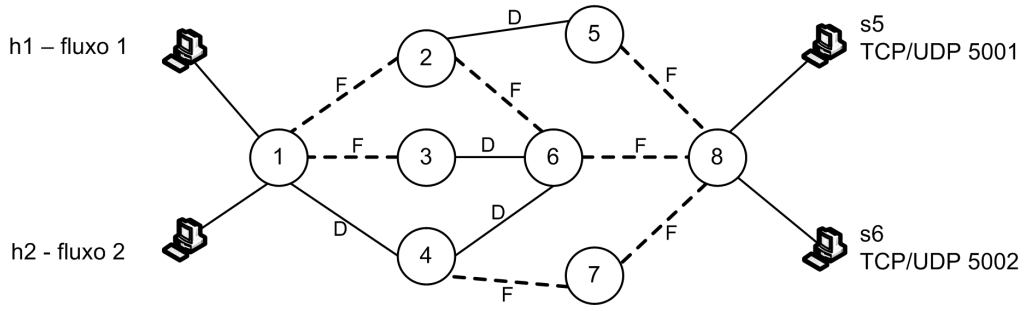
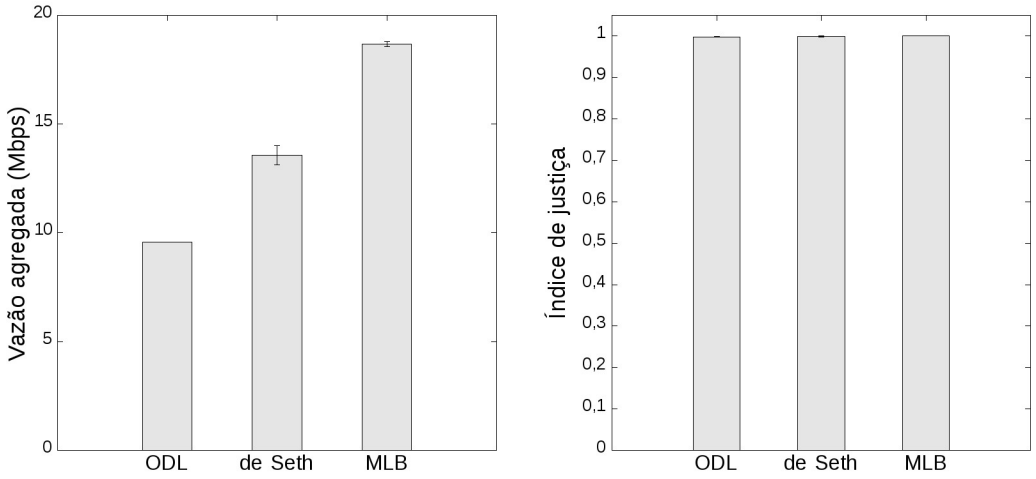


Figura 2. Exemplo de topologia de rede na qual os enlaces tracejados indicam os caminhos padrão designados pela árvore de cobertura gerada pelo ODL.

5.1. Resultados

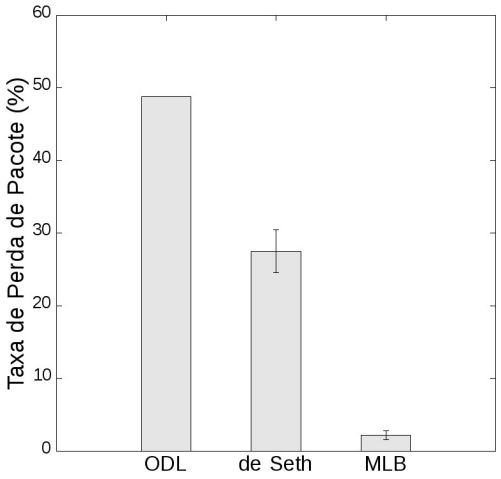
A Figura 3(a) apresenta os resultados da vazão agregada utilizando o modelo de tráfego TCP-2f. Nesta é possível observar que utilizando apenas o ODL, a vazão agregada é de aproximadamente 10 Mbps. Neste caso, o fluxo principal e o fluxo de fundo trafegam juntos dividindo a largura de banda do caminho padrão montado pelo ODL (árvore de cobertura) formado pelos nós 1-2-6-8. Assim, como não existe balanceamento de carga, o valor da vazão agregada será menor ou igual a largura de banda do caminho padrão. No balanceamento de Seth, a vazão agregada apresenta um considerável aumento de 41% em relação ao ODL. Isso porque o balanceamento de Seth realiza a comutação do fluxo principal para um novo caminho alternativo que apresente menor volume de tráfego computado periodicamente. O fluxo de fundo não sofre balanceamento de carga e segue pelo

caminho padrão formado pelos nós 1-2-6-8, já o fluxo principal é comutado ora para um caminho sem enlaces compartilhados pelo fluxo de fundo, formado pelo nós 1-4-7-8, ora para o caminho padrão utilizado pelo fluxo de fundo e ora por caminhos com enlaces compartilhados com o caminho padrão, formados pelos nós 1-2-5-8, 1-3-6-8 e 1-4-6-8. Neste caso, com o balanceamento de carga, a vazão agregada alcançou 13,5 Mbps.



(a) Vazão agregada

(b) Justiça



(c) Perda de pacotes

Figura 3. Vazão agregada, taxa de perda de pacotes e índice de justiça do OLD e dos mecanismos de Seth e MLB.

Por último, no balanceamento MLB é possível atestar que a vazão agregada chegou a 18,6 Mbps, o que significa um ganho de 37% em relação à vazão agregada alcançada por de Seth e 95% em relação à vazão do ODL. Esse melhor desempenho é atribuído ao uso de caminhos com enlaces disjuntos e ao controle de comutação que somente permite a comutação do fluxo principal quando 50% ou mais da largura de banda do seu caminho está ocupada e o novo caminho apresenta uma largura de banda ocupada inferior a 10% ou mais em relação à ocupação da largura de banda do caminho atual. Isso faz com que comutações para novos caminhos com a banda já completamente ocupada não ocorram.

Também evita-se que ocorra comutação para um novo caminho que apresente nível de ocupação semelhante ao do atual caminho em uso.

A Figura 3(b) apresenta os resultados do índice de justiça utilizando o modelo de tráfego TCP-2f. Com o uso apenas do ODL, a disputa pelo uso da largura de banda entre os fluxos resultou em um índice de justiça de 0,99, o que caracteriza que a disputa pela ocupação da banda do caminho padrão ocorreu de maneira semelhante por ambos os fluxos. A taxa média de vazão do fluxo de fundo alcançou 4,8 Mbps contra 4,7 Mbps do fluxo principal. No balanceamento de carga de Seth, o índice de justiça apresentou o mesmo valor de 0,99; o que demonstra que os fluxos ocuparam a banda de forma semelhante, mesmo ocorrendo a comutação de caminho gerada pelo balanceamento de carga. Isso significa que nos momentos em que o fluxo de fundo e o fluxo principal ocuparam juntos a banda do caminho padrão, a vazão dos fluxos se manteve aproximadamente igual e nos momentos em que a comutação do fluxo principal se deu por um caminho alternativo ambos os fluxos ocuparam praticamente a mesma largura de banda. A taxa média de vazão do fluxo de fundo alcançou 6,8 Mbps contra 6,7 Mbps do fluxo principal balanceado. No balanceamento MLB, o resultado do índice de justiça também apresentou o valor de 0,99. A disputa entre os fluxos pela ocupação da largura de banda ocorreu de maneira semelhante. A taxa média de vazão do fluxo de fundo alcançou 9,3 Mbps contra 9,3 Mbps do fluxo principal balanceado. Cabe destacar que mesmo o fluxo de fundo foi beneficiado indiretamente pelo balanceamento de carga ao utilizar exclusivamente em grande parte do tempo o caminho 1-2-6-8.

Por fim, a Figura 3(c) apresenta os resultados da perda de pacotes utilizando o modelo de tráfego UDP-2f. Nesta é possível observar que com o uso do ODL a perda de pacotes chega perto de 50%. Esse alto valor pode ser explicado pelo fato dos fluxos trafegarem juntos pelo caminho padrão formado pelos nós 1-2-6-8. A largura de banda do caminhos padrão é disputada pelos dois fluxos que por serem UDP não têm controle de congestionamento. Assim, os dois hosts de origem transmitem os fluxos a uma taxa constante de 9,5 Mbps, o que leva à grande perda de pacotes. No balanceamento de Seth, a perda de pacotes chega perto de 27%, pois a perda ocorre principalmente quando o fluxo principal balanceado divide o caminho padrão com o fluxo de fundo ou utiliza algum caminho alternativo com enlaces compartilhados pelo caminho padrão, como por exemplo nos casos dos caminhos formados pelos nós 1-2-5-8, 1-3-6-8 e 1-4-6-8. Com a comutação do fluxo principal para o caminho formado pelos nós 1-4-7-8, os fluxos passam a não disputar mais banda entre si e a perda de pacotes não é significativa. No balanceamento MLB a perda de pacotes fica abaixo de 5%. Isso ocorre, porque o controle de comutação somente permite que ocorra a comutação do fluxo principal quando houver uma diferença de 10% ou mais na taxa de ocupação do caminho atual quando comparado ao novo caminho. Quando os caminhos estão com suas larguras de banda ocupadas com níveis de carga parecidos, a comutação não ocorre. Desta forma, no caso em que o fluxo principal segue por um caminho que não compartilhe nenhum enlace com o caminho padrão, como por exemplo o caminho formado pelos nós 1-4-7-8, a ocupação dos caminhos ocorrerá de maneira semelhante. Pois, com a atuação do controle de comutação os fluxos permanecerão por mais tempo seguindo por caminhos distintos sem que ocorra nenhuma comutação. E desta forma, por não existir disputa de banda entre os fluxos, mais pacotes são entregues com sucesso ao seu destino e consequentemente a perda de pacotes diminui. O MLB apresentou uma perda de pacotes 13 vezes menor em relação à perda do mecanismo de

Seth e 24 vezes menor em relação a do ODL.

6. Conclusão

Este trabalho propôs um mecanismo de balanceamento de carga entre caminhos implementado juntamente com um controle de comutação. O mecanismo MLB mostrou-se uma boa solução para evitar a ociosidade de enlaces causada por protocolos como o STP e consequentemente contribuiu para o aumento da largura de banda agregada na rede. Foi realizada uma avaliação de desempenho focada em apresentar os resultados da vazão agregada, da justiça entre fluxos na rede e da perda de pacotes. Foram comparados os mecanismos MLB e de Seth com o uso do ODL puro. O mecanismo MLB apresentou melhores resultados em relação ao modo de funcionamento padrão do ODL e ao mecanismo de Seth. Este melhor desempenho pode ser atribuído ao uso da computação de caminhos com enlaces disjuntos e ao controle de comutação realizado com base na ocupação do caminho pela carga de dados. A utilização de caminhos com enlaces disjuntos pode evitar o compartilhamento da largura de banda pelos fluxos e consequentemente levar à diminuição da perda de pacotes. Já o controle de comutação evita que ocorram comutações de fluxos para caminhos com níveis de ocupação saturados e que possam causar congestionamento na rede ao invés de evitá-lo.

Como uma direção para trabalhos futuros, propõe-se a realização de novos experimentos de balanceamento de carga com o mecanismo MLB em um ambiente real, no qual espera-se fazer uso de comutadores de *hardware* compatíveis com o protocolo OpenFlow, além de utilizar tráfego de dados reais gerados por *hosts* reais.

Um segundo caminho a ser seguido consiste em buscar a utilização da arquitetura SDN de forma distribuída com a utilização de mais de um controlador de rede simultaneamente. Dessa maneira, espera-se tornar a rede mais robusta e tolerante a falhas causadas por situações imprevisíveis, como por exemplo à queda da comunicação entre comutador e controlador de rede ocasionada por uma indisponibilidade no servidor que hospeda o controlador. Além disso, com o uso de mais controladores é possível aumentar a escalabilidade da rede com a inclusão de novos *hosts*. Desta maneira, será possível acompanhar o desempenho do mecanismo MLB em uma rede maior composta por um grande número de comutadores e consequentemente com comunicações entre origem e destino estabelecidas por intermédio de um número maior de saltos.

Referências

Bhandarkar, S. and Khan, K. A. (2015). Load balancing in software-defined network (SDN) based on traffic volume. *Advances in Computer Science and Information Technology (ACSIT)*, 2(7):72–76.

Bredel, M., Bozakov, Z., Barczyk, A., and Newman, H. (2014). Flow-based load balancing in multipathed layer-2 networks using openflow and multipath-TCP. In *Workshop on Hot Topics in Software Defined Networking (HotSDN)*, pages 213–214, New York, NY, USA. ACM.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2012). *Algoritmos Teoria e Prática*. Elsevier, Rio de Janeiro. Tradução da 3ª edição americana.

Costa, L. H. M. K., de Amorim, M. D., Campista, M. E. M., Rubinstein, M. G., Florissi, P., and Duarte, O. C. M. B. (2012). Grandes massas de dados na nuvem: Desafios

e técnicas para inovação. In *Minicurso apresentado no XXX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, SBRC*. SBC.

Hagberg, A. A., Schult, D. A., and Swart, P. J. (2008). Exploring network structure, dynamics, and function using NetworkX. In Varoquaux, G., Vaught, T., and Millman, J., editors, *Proceedings of the 7th Python in Science Conference (SciPy)*, pages 11–15, Pasadena, CA USA.

Hassan, M. H. O. (2017). Implementing Nayan Seth's dynamic load balancing algorithm in software-defined networks: A case study. Dissertação de Mestrado, Sudan University of Science and Technology, Sudan.

Jain, R. (1991). *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. John Wiley & Sons, 1ª edition.

Kreutz, D., Ramos, F. M. V., Veríssimo, P. E., Rothenberg, C. E., Azodolmolky, S., and Uhlig, S. (2015). Software-Defined Networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76.

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

Mallik, A. and Hegde, S. (2014). A novel proposal to effectively combine multipath data forwarding for data center networks with congestion control and load balancing using Software-Defined Networking approach. In *International Conference on Recent Trends in Information Technology (ICRTIT)*, pages 1–7, Chennai, India. IEEE.

OpenDaylight (2017). Opendaylight - The L2 Switch project provides Layer2 switch functionality. https://docs.opendaylight.org/en/stable-fluorine/user-guide/l2switch-user-guide.html. Acesso em: 03 de dezembro de 2019.

Prabhavat, S., Nishiyama, H., Ansari, N., and Kato, N. (2012). On load distribution over multipath networks. *IEEE Communications Surveys & Tutorials*, 14(3):662–680.

Ramdhani, M. F., Hertiana, S. N., and Dirgantara, B. (2016). Multipath routing with load balancing and admission control in software-defined networking (SDN). In *International Conference on Information and Communication Technology (ICoICT)*, pages 1–6, Bandung, Indonesia. IEEE.

Seth, N. (2016). SDN load balancing. https://github.com/nayanseth/sdn-loadbalancing. Acesso em: 16 de abril de 2019.

Singh, S. K., Das, T., and Jukan, A. (2015). A survey on internet multipath routing and provisioning. *IEEE Communications Surveys & Tutorials*, 17(4):2157–2175.

Sun, Z., Xie, Z., Chen, Z., and Dai, L. (2012). An algorithm for the shortest pairs of arc-disjoint paths problem. In *International Conference on Natural Computation (ICNC2012)*, pages 1001–1006, Chongqing, China. IEEE.

Tsai, J. and Moors, T. (2006). A review of multipath routing protocols: From wireless ad hoc to mesh networks. In *ACoRN Early Career Researcher Workshop on Wireless Multihop Networking*, volume 30, Sydney, Australia.