UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

PEDRO OLÍMPIO NOGUEIRA DE OLIVEIRA PINHEIRO

UMA HEURÍSTICA BASEADA EM ARREDONDAMENTO PROBABILÍSTICO PARA ROTEAMENTO EM REDES DATACENTER

QUIXADÁ

2018

PEDRO OLÍMPIO NOGUEIRA DE OLIVEIRA PINHEIRO

UMA HEURÍSTICA BASEADA EM ARREDONDAMENTO PROBABILÍSTICO PARA ROTEAMENTO EM REDES DATACENTER

Monografia apresentada no curso de Ciência da Computação da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Ciência da Computação. Área de concentração: Computação.

Orientador: Prof. Dr. Críston Pereira de Souza

Coorientador: Prof. Me. Jeandro de Mesquita Bezerra

QUIXADÁ

2018

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

P721h Pinheiro, Pedro Olímpio Nogueira de Oliveira.
Uma heurística baseada em Arredondamento Probabilístico para roteamento em redes Datacenter / Pedro Olímpio Nogueira de Oliveira Pinheiro. – 2018.
30 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Ciência da Computação, Quixadá, 2018.
Orientação: Prof. Dr. Críston Pereira de Souza.
Coorientação: Prof. Me. Jeandro de Mesquita Bezerra.

1. Algoritmos. 2. Otimização. 3. Programação Inteira. I. Título.

CDD 004

PEDRO OLÍMPIO NOGUEIRA DE OLIVEIRA PINHEIRO

UMA HEURÍSTICA BASEADA EM ARREDONDAMENTO PROBABILÍSTICO PARA ROTEAMENTO EM REDES DATACENTER

Monografia apresentada no curso de Ciência da Computação da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Ciência da Computação. Área de concentração: Computação.

Aprovada em: ___/___/_____

BANCA EXAMINADORA

Prof. Dr. Críston Pereira de Souza (Orientador)
Universidade Federal do Ceará – UFC

Prof. Me. Jeandro de Mesquita Bezerra (Coorientador)
Universidade Federal do Ceará - UFC

Prof. Dr. Paulo de Tarso Guerra Oliveira
Universidade Federal do Ceará - UFC

Prof. Dr. Wladimir Araujo Tavares
Universidade Federal do Ceará - UFC

AGRADECIMENTOS

Agradeço aos meus irmãos Manoel e Junior, pela amizade sensacional.

Agradeço aos meus irmãos Theo e Ester, pela alegria que trouxeram e trazem.

Agradeço aos meus pais, pelo constante apoio e orientação.

Agradeço à minha família, por estarem sempre comigo.

Agradeço ao meu orientador Críston, pela ajuda, paciência e orientação.

Agradeço ao meu coorientador Jeandro, pela ajuda, paciência e orientação.

Agradeço aos professores Paulo e Wladimir, que além de grandes professores e mestres, são também grandes amigos.

Agradeço a todos os Computeiros do Zodíaco, por todos os momentos que passamos juntos nessa jornada.

Agradeço a todos os meus amigos, por todos os momentos que passamos juntos nessa jornada.

Agradeço a todos que me ajudaram nesse ciclo.

"Você também pode se tornar super-herói."

(All Might)

RESUMO

Atualmente é comum encontrar grandes redes de computadores, com até milhares de dispositivos na mesma rede. Essa quantidade massiva de dispositivos transmitindo informações exige uma alta capacidade de roteamento da rede. Visando melhorar heurísticas já existentes, esse trabalho apresentará uma heurística que, utilizando arredondamento probabilístico, pretende otimizar o processo de roteamento. Para isso, a heurística foi implementada em uma aplicação que emula uma rede e comparada a outras já existentes. Como resultado temos que a heurística, em geral, melhora a eficiência da rede.

Palavras-chave: Roteamento. *Fat-Tree*. Arredondamento Probabilístico. Relaxação Linear.

ABSTRACT

It is now common to find large computer networks, with up to thousands of devices on the same network. This massive amount of devices transmitting information requires a high network routing capability. In order to improve already existing heuristics, this work will present a heuristic, which, using probabilistic rounding, intends to optimize the routing process. For this, the heuristic was implemented in an application that emulates a network and compared to other already existing ones. As a result we have that heuristics, in general, improves the efficiency of the network.

Keywords: Routing. Fat-Tree. Probabilistic Rounding. Linear relaxation.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

SUMÁRIO

1 INTRODUÇÃO

O grande crescimento no tamanho das redes de computadores dificulta o gerenciamento e roteamento dessas. Com a existência de até dezenas de milhares de máquinas em uma mesma rede, estabelecer uma boa comunicação entre qualquer par de computadores pode ser um problema difícil de resolver, principalmente quando necessário transmitir grande quantidade de dados.

Em *datacenters* atuais, segundo Wang et al. (2014), é muito comum utilizar topologias que permitem os *switches* escolherem rotas diferentes para demandas por fluxo de dados com destinos finais iguais. Como os *links* entre os *switches* possuem uma largura de banda finita, grandes demandas de fluxo podem os sobrecarregar, o que é desejável que não ocorra, pois pode ocasionar perda de dados e atraso em diversas aplicações.

O problema de selecionar as rotas para as demandas por fluxo pode ser modelado como um problema próximo do *multicommodity flow*, que é um problema NP-Completo, segundo Even, Itai e Shamir (1975). Neste trabalho será apresentado uma heurística que, utilizando arredondamento probabilístico busca obter uma solução que se aproxima da ótima. Essa heurística propõe minimizar a maior sobrecarga dos *links*. Como resultado da heurística proposta pretende-se obter um Roteamento em *Fat-Tree* (RFT) mais balanceado, com as demandas de fluxo melhor divididas entre os *links*.

O RFT consiste em, dado um grafo $G = (V, E)$ em que as arestas $(u, v) \in E$ possui uma capacidade $c(u, v) \geq 0$ e um conjunto de fluxos (s, t, d) é necessário, para cada fluxo, encontrar caminhos no grafo que comecem em s, terminem em t e utiliza d da capacidade das arestas obedecendo as restrições:

1. **Capacidade dos Link**: As aresta (u, v) deve ter um conjunto de *commodities* nela alocada que supere sua capacidade $c(u, v)$ no mínimo possível, sem superar quando viável.
2. **Conservação de fluxo nos vértices de transição**: Para cada *commodity*, cada vértice, que não seja a origem ou o destino da *commodity*, deve ter uma quantidade igual de fluxo dessa *commodity* K_i entrando e saindo.
3. **Conservação de fluxo nos vértices de origem**: Os vértices de origem de uma *commodity* possui uma fluxo de saída superior ao de entrada em 100%, pois ali 100% do fluxo é gerado.
4. **Conservação de fluxo nos vértices de destino**: Os vértices de destino de uma *commodity* possui uma fluxo de entrada superior ao de saída em 100%, pois ali será consumido 100%

do fluxo.

Este trabalho terá aplicação em redes de *datacenters* com uma topologia específica: as *fat-trees*, árvores multi-raizadas em que os dispositivos finais (servidores) estão sempre nas folhas.

O objetivo desse trabalho é propor uma heurística para roteamento para uma rede *datacenter* com SDN (*Software Defined Network* - Redes Definidas por *Software*) com topologia *fat-tree* definido por *software*.

Para avaliação da heurística proposta, será utilizada uma aplicação que emula uma rede *datacenter* com a topologia descrita e o processo de roteamento, comparando os resultados obtidos com outras heurísticas amplamente utilizadas: Hedera e ECMP (AL-FARES et al., 2010). O ECMP utiliza *hashs* estáticos para determinar caminhos para os fluxos. Porém esses *hashs* não consideram a rede atual ou o tamanho do fluxo para tomada de decisões, resultando em sobrecarga de *switches* e *links*.

Esse trabalho está estruturado da seguinte forma: no Capítulo 2 estão descritos os principais conceitos utilizados no trabalho. No Capítulo 3 são apresentados os trabalhos de Al-Fares et al. (2010) e Qian, Hu e Yeung (2016). No Capítulo 4 são descritos os passos utilizados para realização desse trabalho. O Capítulo 5 mostra os resultados obtidos. No Capítulo 6 são apresentadas as conclusões.

2 FUNDAMENTAÇÃO TEÓRICA

Nesse capítulo serão apresentados os principais conceitos que fundamentam o trabalho.

2.1 Redes Definidas por *Software*

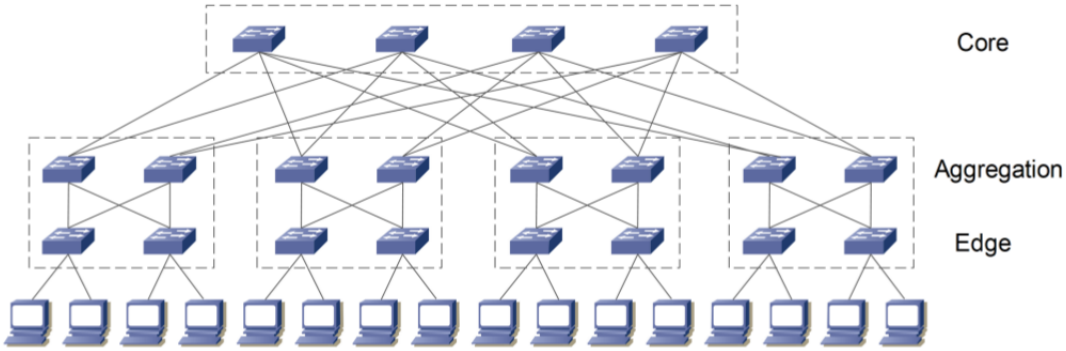
Segundo Santos et al. (2018) Redes Definidas por Software (SDN) como uma alternativa para superar problemas enfrentados nos ambientes de redes de computadores. A grande inovação se baseia na dissociação do plano de controle do plano de dados. Dessa forma, *switches* tem seu plano de dados gerenciado por um controlador SDN (entidade que age como o plano de controle). Consequentemente, ao contrário das redes tradicionais, esses *switches* não implementam o plano de controle e as decisões de encaminhamento são baseadas em fluxos encaminhados do *switch* para o controlador.

2.2 Fat-tree

Segundo Al-Fares et al. (2010) *Fat-tree*, é uma topologia de rede muito comum nos *datacenters* atuais. Proposto por Al-Fares, Loukissas e Vahdat (2008), consiste em uma árvore multi-enraizada, em que todos os servidores (dispositivos finais) estão em folhas e que existem diversas rotas possíveis para conectar qualquer par de servidores. A Figura 1 representa uma *fat-tree*. Uma *fat-tree* pode ter vários níveis, geralmente 2 ou 3, segundo Wang et al. (2014). A *fat-tree* que utilizaremos contém 3 níveis: *Core*, *Aggregation*, *Edge*. Todos os *links* da rede são capazes de transmitir dados nas duas direções.

Os *switches* da camada *Edge* são os *switches* que estão conectados nos dispositivos finais, ou servidores (*switches* mais inferiores na Figura 1). Eles também se conectam a *switches* da camada *Aggregation*, porém somente nos *switches* que estão no mesmo *pod*. Os *pods* são conjuntos de *switches* da camada *Edge* e *Aggregation*, representados na Figura 1 pelos quadrados tracejados. Os *pods* não possuem *links* diretos entre si, porém estão conectados aos *switches* da camada *Core*. Cada *switches* da camada *Core* possui *links* para *switches* da camada *Aggregation* em todos os *pods*.

Figura 1 – *4-Fat-Tree* com 3 níveis



Fonte – Wang et al. (2014).

Uma rede com topologia *fat-tree* é construída baseada no número de portas que *switches* possuem, considerando que todos os *switches* possuem o mesmo número de portas. Cada servidor está conectado a apenas um *switch* da camada *Edge* e somente a esse.

Uma *k-fat-tree* é a *fat-tree* onde todos os *switches* possuem k portas. Sua estrutura é construída por k *pods* (caixas com 4 *switches* na Figura 1). Cada *pod* contém k *switches*. Desses, $k/2$ são da camada *Edge* (camada de *switches* inferior) e $k/2$ são da camada *Aggregation* (camada de *switches* central). Cada *switch* da *Edge* tem $k/2$ de suas portas se comunicando com os servidores e as outras $k/2$ com todos os *switches* da *Aggregation* que estão no mesmo *pod*. Os *switches* da *Aggregation* se conectam aos $k/2$ *switches* da camada *Edge* que estão em seu *pod* e à $k/2$ *switches* da camada *Core* (camada de *switches* superior), distintos dos que os outros *switches* do mesmo *pod* se conectam (na Figura 1, no primeiro *pod*, o primeiro *switch* da *Aggregation* se conecta aos dois primeiros da *Core* e o segundo da *Aggregation* se conecta aos dois últimos da *Core*). Cada *switch* da camada da camada *Core* possui uma conexão à k *switches* da *Aggregation*, em k *pods* diferentes.

Como existem $k/2$ *switches* da *Aggregation* que se conectam à $k/2$ *switches* diferentes na camada *Core*, existem ao todo $(k/2)^2$ *switches* na camada *Core*. Como existem k *pods* e $k/2$ *Aggregation* mais $k/2$ *Edge* por *pod*, em toda a rede existem $k^2/2$ *switches* na camada *Aggregation* e $k^2/2$ *switches* na camada *Edge*.

A heurística apresentada será aplicada exclusivamente em redes datacenter com essa topologia.

2.3 Programação linear

No problema geral de Programação Linear (PL), desejamos otimizar uma função linear de acordo com um conjunto de desigualdades lineares.

Dado um conjunto de números reais $a_1, a_2, \ldots, a_n$ e um conjunto de variáveis $x_1, x_2, \ldots, x_n$, uma função linear f sobre essas variáveis é definida por:

$$f(x_1, x_2, \ldots, x_n) = a_1x_1 + a_2x_2 + \cdots + a_3x_3 = \sum_{j=1}^{n} a_jx_j \tag{2.1}$$

Se b é um número real e f uma função linear, então a equação:

$$f(x_1, x_2, \ldots, x_n) = b \tag{2.2}$$

é uma igualdade linear e as desigualdades

$$f(x_1, x_2, \ldots, x_n) \leq b \tag{2.3}$$

e

$$f(x_1, x_2, \ldots, x_n) \geq b \tag{2.4}$$

são desigualdades lineares. (CORMEN, 2009)

Por exemplo:

$$\text{maximize } 2x_1 + x_2 \tag{2.5}$$

sujeito a:

$$x_1 + x_2 \leq 6 \tag{2.6}$$

$$x_1 \leq 3 \tag{2.7}$$

$$x_1, x_2 \geq 0 \tag{2.8}$$

Nesse exemplo é necessário encontrar valores para x_1 e x_2, ambos maiores ou iguais que zero, pela regra da Desigualdade 2.8. Porém x_1 não pode ter um valor superior à 3, pela regra da Desigualdade 2.7. Por fim, a soma de x_1 e x_2 deverá ser inferior ou igual a 6, pela regra da Desigualdade 2.6. Cumprindo essas regras, os valores obtidos devem ter como resultado para a função $2x_1 + x_2$ o maior valor possível. Para esse exemplo a solução é $x_1 = 3$ e $x_2 = 3$.

No problema abordado, a função que será otimizada (minimizada) será um valor que equivale à sobrecarga do *link* com maior sobrecarga. Dessa maneira pretende-se reduzir o maior excesso entre todos os *links*, ou, em caso de excesso negativo (folga), maximizar a menor folga entre todos os *links*. As desigualdades garantem a satisfação das regras do *multicommodity flow*.

2.4 Multicommodity flow

A *fat-tree* em que a heurística é aplicada é modelada como um grafo, para adaptar o problema de selecionar rotas para tráfego de dados para um problema que se aproxima do *multicommodity flow*. Nessa modelagem o *switches* serão os vértices e os *links* as arestas.

Um grafo é um par ordenado (V,E), onde V é um conjunto de vértices e $E \subseteq V \times V$ é um conjunto de arestas que ligam pares de vértices entre si.

O problema de *multicommodity flow* é definido sobre um grafo $G = (V,E)$ onde cada aresta $(u,v) \in E$ possui uma capacidade $c(u,v) \geq 0$ de transportar *commodities*. Uma *commodity* K é definida por uma tupla (s,t,d), onde s é o vértice de origem da commodity, t o destino e d a largura da demanda por fluxo dessa commodity.

Seja $f_i(u,v) \in [0,1]$ a variável que informa quanto da *commodity* K_i passa pela aresta (u,v), variando de 0, se nenhuma fração dela é atendida pela aresta, até 1, se é atendida completamente pela aresta.

A escolha das arestas a que serão atribuídas as *commodities* deve obedecer as seguintes regras:

1. **Capacidade dos Link**: Nenhuma aresta (u,v) deve ter um conjunto de *commodities* $\{K_1, K_2, \ldots, K_k\}$ nela alocada que supere sua capacidade $c(u,v)$. Ou seja:
$$\sum_{i=1}^{k} f_i(u,v)d_i \leq c(u,v), \forall (u,v) \in E \tag{2.9}$$

2. **Conservação de fluxo nos vértices de transição**: Para cada *commodity*, cada vértice, que não seja a origem ou o destino da *commodity*, deve ter uma quantidade igual de fluxo dessa *commodity* K_i entrando e saindo. Ou seja:
$$\sum_{w \in V} f_i(w,u) = \sum_{w \in V} f_i(u,w), \forall i \in \{1,2,\ldots,k\} \tag{2.10}$$

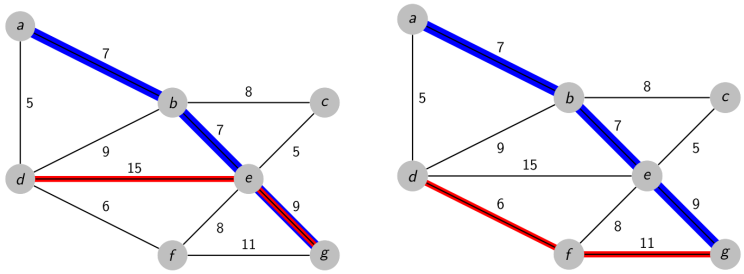
3. **Conservação de fluxo nos vértices de origem**: Os vértices de origem de uma *commodity* possui uma fluxo de saída superior ao de entrada em 1, pois ali 100% do fluxo é gerado:
$$\sum_{w \in V} f_i(s_i,w) - \sum_{w \in V} f_i(w,s_i) = 1, \forall i \in \{1,2,\ldots,k\} \tag{2.11}$$

4. **Conservação de fluxo nos vértices de destino**: Os vértices de destino de uma *commodity* possui uma fluxo de entrada superior ao de saída em 1, pois ali será consumido 100% do fluxo:

$$\sum_{w \in V} f_i(w,t_i) - \sum_{w \in V} f_i(t_i,w) = 1, \forall i \in \{1,2,\ldots,k\} \tag{2.12}$$

A Figura 2 ilustra um grafo, onde os círculos são os vértices e as linhas os conectando são as arestas, e os números próximos as aresta são suas capacidades. Nesse grafos existem dois fluxos: fluxo vermelho, com origem em d, destino em g e tamanho 2, e o fluxo azul, com origem em a, destino em g e tamanho 7. São apresentadas duas maneira válidas de alocar esses fluxos na Figura 2a e na Figura 2b. Note que na solução da Figura 2a a aresta que liga o vértice e ao vértice g está saturada, pois possui capacidade 9 e está sendo utilizada por dois fluxos, de tamanho 2 e 7.

Figura 2 – Exemplo de Multicommodity flow



Fonte – Elaborado pelo autor

O RFT (Roteamento em *Fat-Tree*) não será modelado como *multicommodity flow* porque no RFT pode haver algum *link* que sua capacidade seja superada pelas largura das demandas das *commodities* que utilizam esse *link*. Será utilizado uma variação do problema do *multicommodity flow* para permitir sobrecarga de fluxos nas arestas.

2.5 Arredondamento probabilístico

Encontrar a solução ótima para um modelo de PL com variáveis de decisões, que possuem valor 0 ou 1 para representar uma escolha, para o problema do RFT é NP-Completo (RAGHAVAN; TOMPSON, 1987). Substituindo as variáveis de decisões inteiras por variáveis reais, que representam probabilidades de escolha, obtemos um problema que possui algoritmos conhecidos que o resolvem em tempo polinomial. Essa técnica é conhecida como Arredondamento Probabilístico (WILLIAMSON; SHMOYS, 2011).

Arredondamento Probabilístico é explicado por Williamson e Shmoys (2011) utilizando o exemplo do problema da Cobertura Mínima por Conjuntos.

O problema da Cobertura Mínima por Conjuntos consiste em, dado um conjunto de elementos $E = \{e_1, e_2, \ldots, e_n\}$ e subconjuntos $S_1, S_2, \ldots, S_m \subseteq E$ com pesos $w_1, w_2, \ldots, w_m$, encontrar uma coleção de subconjuntos, que a união contenha todos os elementos de E e que tenha o menor peso possível.

O problema pode ser resolvido utilizando programação linear inteira (caso específico de programação linear, onde as variáveis são inteiras) com a seguinte modelagem:

$$\text{minimize} \sum_{j=1}^{m} w_j x_j \tag{2.13}$$

sujeito a:

$$\sum_{j:e_i \in S_j} x_j \geq 1 \qquad i = 1, \ldots, n \tag{2.14}$$

$$x_j \in \{0,1\} \qquad j = 1, \ldots, m \tag{2.15}$$

A Desigualdade 2.14 garante que, para cada elemento $e_i \in E$, pelo menos um conjunto S_j, que contém e_i, estará na solução.

Nesse modelo, a variável x_j valem 1 se S_j está na solução. Porém, encontrar a solução ótima para esse modelo é um problema NP-Completo. Realizando uma relaxação linear nesse modelo, ao substituir a Equação 2.15 pela Desigualdade 2.16, obtemos um modelo de programação linear com algoritmos eficientes conhecidos para resolução.

$$x_j \geq 0 \qquad j = 1, \ldots, m \tag{2.16}$$

Porém, não existem mais as variáveis inteiras x_j para informar quais os conjuntos que estão na solução. Ao invés teremos x_j nos reais informando a probabilidade de S_j estar na solução ótima. Então é realizado um sorteio com base nas probabilidades obtidas e é atingido uma solução próxima à ótima, onde cada conjunto j é inserido na solução com probabilidade x_j. O sorteio é repetido enquanto não satisfaz a condição de conter todos os elementos de E. Ao final obtemos uma coleção de conjuntos sorteados com base em probabilidades que minimizam a soma dos seus pesos (conjuntos mais leves tem mais chances de estar na solução).

3 TRABALHOS RELACIONADOS

Serão apresentados dois trabalhos que apresentam soluções para o problema do Roteamento em *Fat-Tree* (RFT):

- Hedera: Dynamic Flow Scheduling for Data Center Networks de Al-Fares et al. (2010);
- An Efficient Routing Algorithm in Fat-tree Data Center Networks de Qian, Hu e Yeung (2016).

Al-Fares et al. (2010) apresentam uma nova heurística para roteamento das demandas de fluxo para *datacenters* com topologia *fat-tree* e redes definidas por *software*, o Hedera, e o compara com outras heurísticas já existentes.

Descrito em alto nível, Hedera tem um laço de controle com três passos básicos. Primeiro detecta grandes fluxos de dados em *switches* de borda, que são os que estão conectados diretamente nos dispositivos finais. Depois estima a demanda de grandes fluxos e usa algoritmos de atribuição para computar boas rotas para esses fluxos. Por fim, as rotas são fixadas nos *switches* (AL-FARES et al., 2010).

O Hedera utilizada a heurística *Global First Fit* em que, para cada *switch*, o primeiro *link* disponível que tenha largura de banda suficiente para suportar o tráfego será utilizado por essa demanda. Outra solução citada por Al-Fares et al. (2010) é a *Simulated Annealing*, que realiza uma busca probabilística para selecionar a melhor rota para a demanda.

O ECMP também é utilizado por Al-Fares et al. (2010) para comparar com o Hedera.

Qian, Hu e Yeung (2016) propõem um novo algoritmo de roteamento: o *Global Round Robin* (GRR). A topologia dos *datacenters* para a aplicação do GRR deve ser *fat-tree*. O processo de roteamento é dividido em duas partes: *upstream*, que consiste em transmitir o pacote de dados do emissor até o *switch* que seja o ancestral comum entre o emissor e o receptor mais próximo, e *downstream*, que reencaminha esses pacotes para o destino.

O GRR trabalha principalmente nas decisões necessárias durante o *upstream*, em que além de encontrar o ancestral adequado, seleciona a rota a ser utilizada. Durante o *downstream* os próprios *switches* decidem para qual rota encaminhar os pacotes.

A heurística descrita no atual trabalho seleciona, através de um controlador, todas os *links* a serem utilizados para transmissão de um pacote de dados, não somente no *upstream*. Na heurística apresentada no atual trabalho não existe a necessidade da busca pelo ancestral comum mais próximo da origem e do destino, pois permite o controlador selecionar rotas que passem por ancestrais mais distantes, pois o caminho mais curto pode estar congestionado.

4 PROCEDIMENTOS METODOLÓGICOS

Nesse capítulo será descrito os passos realizados para realização desse trabalho.

4.1 Heurística APR

Inicialmente, apresentamos uma heurística, que utilizando arredondamento probabilístico para aplicação de um modelo de PL (Programação Linear), encontra uma solução para RFT (Roteamento em Fat-Tree). A heurística apresentada será tratada por APR (Arredondamento Probabilístico para Roteamento).

A heurística consiste em duas partes principais: obtenção das probabilidades de escolha dos *links* através das variáveis da PL e o sorteio dos *links* a serem utilizados, utilizando as probabilidades obtidas.

Será criado um modelo Relaxação do *Multicommodity Flow* Adaptado (RMFA) para a primeira parte da heurística:

$$\text{minimize } y \tag{4.1}$$

sujeito a

$$\sum_{i=1}^{k} f_i(u,v)d_i - y \leq c(u,v) \qquad \forall (u,v) \in E \tag{4.2}$$

$$\sum_{w \in V} f_i(u,w) - \sum_{w \in V} f_i(w,u) = 0 \qquad \forall u \in V - \{s_i,t_i\}, \forall i \in \{1,\ldots,k\} \tag{4.3}$$

$$\sum_{w \in V} f_i(s_i,w) - \sum_{w \in V} f_i(w,s_i) = 1 \qquad \forall i \in \{1,\ldots,k\} \tag{4.4}$$

$$\sum_{w \in V} f_i(w,t_i) - \sum_{w \in V} f_i(t_i,w) = 1 \qquad \forall i \in \{1,\ldots,k\} \tag{4.5}$$

$$f_i(u,v) \geq 0 \qquad \forall (u,v) \in E, \forall i \in \{1,\ldots,k\} \tag{4.6}$$

A Desigualdade 4.2 aproxima Equação 2.9 do *multicommodity flow*, mas ela permite a existência de um excesso y na capacidade dos links. Porém a Função Objetivo 4.1 tende a minimizar y. Logo sobrecargas são permitidas, porém evitadas e reduzidas. A Função 4.1 estima o maior excesso de largura entre os *links*. As Equações 4.3, 4.4 e 4.5 garantem que as Equações 2.10, 2.11 e 2.12 do *multicommodity flow* são respeitadas.

Após a execução de um algoritmo para resolução desse modelo de PL é obtido em $f_i(u,v)$, frações dos fluxo da demanda K_i que seria alocada para o *link* (u,v), que serão utilizadas como probabilidades da *commodity*. Então, para cada *commodity* i serão realizados sorteios, iniciando de s_i, para obter o próximo vértice a ser utilizado na rota, até que alcance t_i.

O Algoritmo 1 descreve como o sorteio é realizado. Os argumentos são: V é a quantidade de vértices, p é uma matriz, que é preenchida com a solução da relaxação linear da RMFA, em que $p[a][b]$ é um valor entre 0 e 1, indicando a chance da demanda ser encaminhada para b quando alcança a, i é o vértice em que devemos sortear a próxima aresta que a demanda será encaminhada e $destination$ o destino da demanda. No início do algoritmo, verifica-se se o destino já foi alcançado, em caso positivo, devolve uma lista de vértices contendo apenas o destino. Em caso negativo, é criado um vetor *chance*, no qual cada elemento de *chance* é uma tupla (d,c), em que c é o acumulado de probabilidades dos elementos anteriores à c serem escolhidos mais a probabilidade de d ser escolhido. Dessa maneira d será o vértice seguinte se o valor sorteado s for maior que o acumulado e menor que o acumulado mais a probabilidade de d. Então é sorteado um valor s e procurado em *chance* o valor de d adequado. É feita uma chamada recursiva para determinar quais os próximos vértices utilizados. Então é adicionado o vértice i e devolvida a rota.

Algoritmo 1: Sorteio

```
Function Sortear(V, p, i, destination):
    if i = destination then
        r ← [i]
        return r
    end
    chance ← []
    for j ← 1 to V do
        if p[i][j] > 0 then
            Inserir no fim(chance, (j, Último(chance).y + p[i][j]))
        end
    end
    s ← Aleatório entre(0, 1)
    j ← 1
    while j ≤ Tamanho(chance) − 1 e s > chance[j].y do
        j ← j + 1
    end
    path ← Sortear(V, p, chance[j].x, destination)
    Inserir na frente(path, i)
    return path
```

Fonte - Elaborado pelo autor.

Por exemplo, se o vértice 1 tem 20% de chance de enviar para o vértice 2, 0% para o vértice 3, 50% para o vértice 4 e 30% para o vértice 5. O vetor *chance* será construído da seguinte forma: $[(2,0.2),(4,0.7),(5,1)]$. Então, caso $s = 0.1$, o vértice sorteado será o 2, mas caso $s = 0.5$, o vértice sorteado será 4.

O APR não força os fluxos a serem encaminhados pelos caminhos mais curtos, podendo optar por uma rota mais comprida, porém menos sobrecarregada. Por exemplo, em um caso que alguns *links* que ligam os *switches* da *Aggregation* com os *switches* da *Edge* estão sobrecarregados, existe um caminho que passa por um *switch* na camada *Core*, entra em um *pod* diferente, retorna à camada *Core* em um *switch* diferente e entra no *pod* original do fluxo em um *switch* da *Aggregation* diferente que possui um *link* sem sobrecarga para o *switch* da *Edge* que é o destino do fluxo.

4.2 Ambiente de Avaliação

Essa seção descreverá o ambiente utilizado para avaliação do APR.

O APR foi implementado na linguagem C++, utilizando o otimizador linear CPLEX para resolver o modelo de PL do RMFA (IBM, 2018).

A heurística foi integrada a uma aplicação que emula uma rede *datacenter* com topologia *Fat-Tree* utilizando o *mininet* (HANDIGOL et al., 2012), e o controlador de redes definidas por *software Ryu*. A aplicação foi desenvolvida por Huagmachi (2018).

Para a integração, na função que recebia as demandas como parâmetros, escolhia as rotas de acordo com a heurística utilizada e alocava essas rotas para essas demandas, a heurística foi substituída pela implementação do APR descrita.

A máquina utilizada para execução da aplicação utiliza o sistema operacional Ubuntu 16.04.5 LTS, com um processador Intel(R) Xeon(R) CPU 5506 @ 2,13GHz 64bits e 24GB de memória RAM.

Serão utilizadas topologias do tipo *4-fat-trees*, que suporta até 16 servidores na rede, e *8-fat-trees*, que suporta até 128 servidores.

Os tráfegos gerados seguem um padrão, por exemplo: no padrão stag_0.5_0.3, o 0.5 significa que 50% do fluxo sob o mesmo *switch* da *Edge*, o 0.3 que 30% sob diferentes *switches*

da *Edge* do mesmo *pod* e os 20% restantes entre *pods* diferentes. Todos os experimentos foram realizados com 4 padrões de tráfegos diferentes:

Tabela 1 – Descrição dos cenários de experimentos

Padrão	Fluxo sob mesmo switch da Edge	Fluxo sob switch Edge diferente no mesmo pod	Fluxo em pods diferentes
stag_0.5_0.3	50%	30%	20%
stag_0.6_0.2	60%	20%	20%
stag_0.7_0.2	70%	20%	10%
stag_0.8_0.1	80%	10%	10%

Fonte – Elaborado pelo autor

4.3 Avaliação de Performance

Assim como no trabalho de Zhang, Cui e Zhang (2017), o APR será comparado com ECMP e Hedera (AL-FARES et al., 2010).

Os critérios utilizados para avaliação serão:

- **Taxa de Transferência Normalizada**: Total da transferência combinada de todos os servidores dividida pela soma da capacidade total dos *links* dos servidores.
- **Atraso Médio de Ida e Volta**: Atraso médio para transmissão de pacote de dados (em milissegundos).
- **Taxa de perda de pacotes**: Taxa de perda de pacotes.

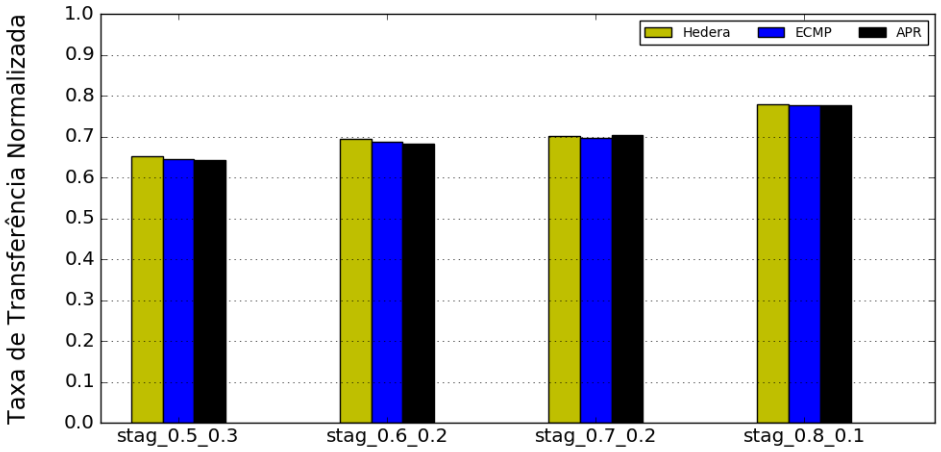
A função objetivo (Equação 4.1) visa diminuir excesso de carga em um mesmo *link*, evitando congestionamento e melhorando o desempenho da rede, portanto gerando efeito direto sobre os critérios utilizados para comparação.

5 RESULTADOS

Os resultados obtidos após a execução dos experimentos são apresentados em gráficos. Os gráficos das figuras 3, 4 e 5 se referem aos experimentos realizados com topologias do tipo *4-fat-tree*. Os gráficos das Figuras 6, 7 e 8 se referem aos experimentos realizados com topologias do tipo *8-fat-tree*.

O gráfico da Figura 3 mostra, para cada padrão de tráfego gerado nos experimentos, o quanto da capacidade total da rede cada algoritmo utilizou. Apesar do uso da largura da rede aumentar conforme o tráfego se concentra sob o mesmo *switch* da camada *Edge*, as heurísticas não apresentaram grandes diferenças entre si.

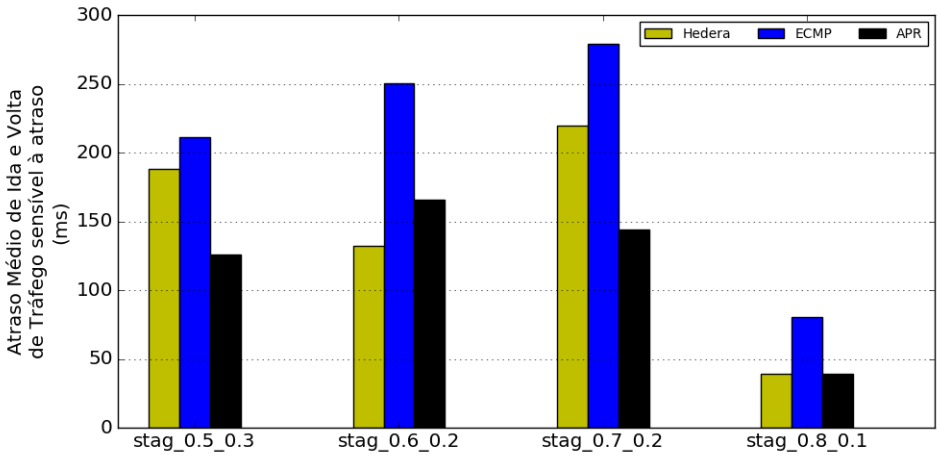
Figura 3 – Taxa de Transferência Normalizada com k = 4



Fonte – Elaborado pelo autor

O gráfico da Figura 4 mostra, também para cada padrão de tráfego gerado, o atraso médio para ida e volta da fonte para o destino dos pacotes. Comparado ao ECMP, o APR apresentou uma redução no tempo médio de ida e volta dos pacotes, chegando a ser até 100 milissegundos mais rápido (aproximadamente 30% menos tempo) para o tráfego stag_0.7_0.2. Comparado ao Hedera, o APR obteve um atraso médio maior para o padrão de tráfego stag_0.6_0.2, aproximadamente 20 milissegundos mais demorado, obteve atraso médio igual para o padrão de tráfego stag_0.8_0.1 e um atraso médio menor para os dois padrões restantes, de 50 à 70 milissegundos mais rápido (25 à 30% menos tempo).

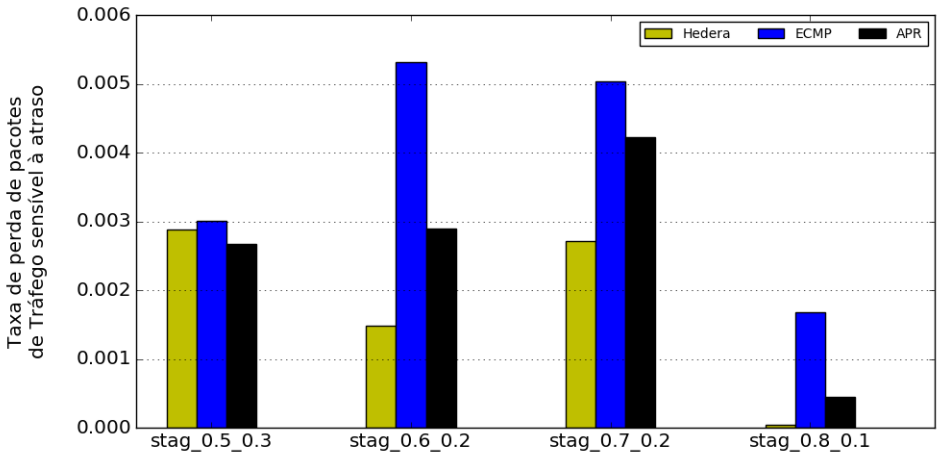
Figura 4 – Desvio padrão para Atraso de Ida e Volta com k = 4



Fonte – Elaborado pelo autor

O gráfico da Figura 5 mostra a taxa de perda de pacotes. Para esse critério, o APR também superou o ECMP em todos os padrões de tráfego. Comparado ao Hedera, a perda de pacotes aumenta em três dos padrões de tráfego, em no máximo 0.15%, e reduziu em um (stag_0.5_0.3).

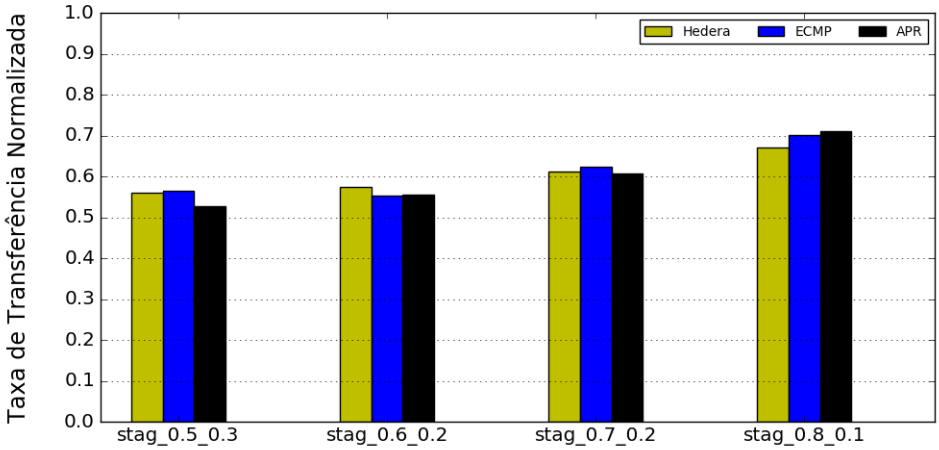
Figura 5 – Taxa de perda de pacotes com k = 4



Fonte – Elaborado pelo autor

O gráfico da Figura 6 mostra que, mesmo com o acréscimo do número de nós na rede, a taxa de uso da largura da banda continuou semelhante para as três heurísticas avaliadas.

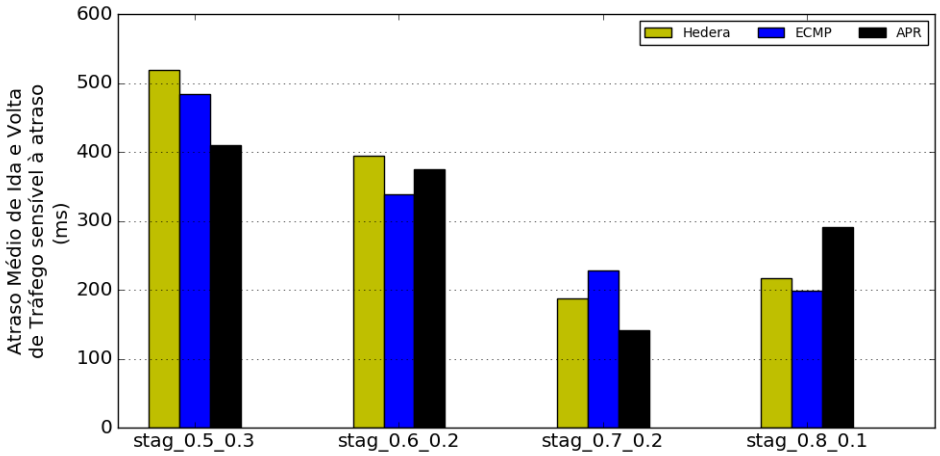
Figura 6 – Taxa de Transferência Normalizada com k = 8



Fonte – Elaborado pelo autor

Considerando o atraso para ida e volta dos pacotes, descrito no gráfico da Figura 7, o APR vence o Hedera em três dos quatro padrões de tráfego utilizados no experimento, demorando, em média, 100 milissegundos a menos para o caso do tráfego stag_0.5_0.3. Nesse mesmo critério, comparado ao ECMP, o APR é mais rápido em média na metade dos tráfegos, variando de aproximadamente 100 milissegundos mais lenta até 100 milissegundos mais rápido.

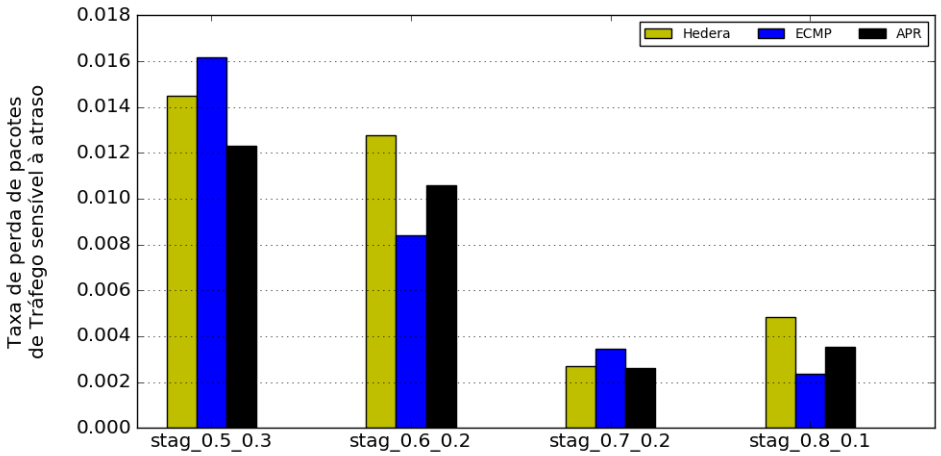
Figura 7 – Atraso Médio de Ida e Volta com k = 8



Fonte – Elaborado pelo autor

O gráfico da Figura 8, mostra que, nos experimentos realizados, o Hedera sempre tem uma perda de pacotes maior que o APR, enquanto o ECMP tem uma perda menor do que o APR para os padrões de tráfego stag_0.6_0.2 e stag_0.8_0.1, porém no máximo 0.2% a menos.

Figura 8 – Taxa de perda de pacotes com k = 8



Fonte – Elaborado pelo autor

Quando ocorre colisões no *hash* utilizado pelo ECMP, ele tende a encaminhar pacotes por caminhos sobrecarregados, ocasionando congestionamento e atrasos na transmissão. Hedera foca em atribuir um fluxo pelo melhor caminho, mas ignora que esse caminho pode ter sido atribuído para outro fluxo, diminuindo a performance da rede (ZHANG; CUI; ZHANG, 2017). O APR evita enviar fluxo por caminhos já ocupados, evitando os conflitos de caminhos causados pelas outras heurísticas e melhorando a performance da rede.

6 CONSIDERAÇÕES FINAIS

Esse trabalho apresentou APR, uma heurística para roteamento em redes *datacenter* com topologia *Fat-Tree*. APR utiliza relaxação linear com arredondamento probabilístico para rotear grandes pacotes de dados, visando distribuir mais uniformemente por toda a largura da rede, evitando sobrecarga dos *links*.

Para avaliação da heurística descrita foram executados experimentos em que o APR foi executado em paralelo com duas heurísticas amplamente utilizadas: ECMP e Hedera (AL-FARES et al., 2010). Para execução dos experimentos, foi utilizada uma rede emulada pelo *mininet* e um controlador *Ryu* (HUAGMACHI, 2018). Dos experimentos constata-se que o APR pode efetivamente otimizar a performance da rede, utilizando melhor sua total capacidade, melhorando o desempenho de diversas aplicações, como sistemas web, banco de dados, *hadoop* entre outros.

Com o acréscimo do número de *switches* na rede, quando muda o valor de k de 4 para 8, o APR manteve-se eficiente, mostrando boa escalabilidade da heurística.

Com base nos gráficos conclui-se que com o acréscimo de tráfego que necessita passar pelos *switches* da camada *Core* o APR obtém resultados melhores em comparação com as outras heurísticas.

As principais dificuldades foram integrar a heurística na aplicação que emula a rede, pois é uma aplicação grande e complexa, e executar experimentos para redes com outras topologias (diferente valores de k) e outras largura de banda nos *links*.

As sugestões para trabalhos futuros são testar a heurística com outros padrões de tráfego ou com tráfego real, variar o valor de *k* (que define a topologia da rede) e a largura de banda dos *links* e avaliar o tempo de execução do otimizador, que não foi possível nesse trabalho pela dificuldade de separar o a execução do otimizador da execução do emulador.

REFERÊNCIAS

AL-FARES, M.; LOUKISSAS, A.; VAHDAT, A. A scalable, commodity data center network architecture. In: ACM. **ACM SIGCOMM Computer Communication Review**. [S.l.], 2008. v. 38, n. 4, p. 63–74.

AL-FARES, M.; RADHAKRISHNAN, S.; RAGHAVAN, B.; HUANG, N.; VAHDAT, A. Hedera: Dynamic flow scheduling for data center networks. In: **Nsdi**. [S.l.: s.n.], 2010. v. 10, p. 19–19.

CORMEN, T. H. **Introduction to algorithms**. [S.l.]: MIT press, 2009.

EVEN, S.; ITAI, A.; SHAMIR, A. On the complexity of time table and multi-commodity flow problems. In: IEEE. **Foundations of Computer Science, 1975, 16th Annual Symposium on**. [S.l.], 1975. p. 184–193.

GARG, N.; KOENEMANN, J. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. **SIAM Journal on Computing**, SIAM, v. 37, n. 2, p. 630–652, 2007.

HANDIGOL, N.; HELLER, B.; JEYAKUMAR, V.; LANTZ, B.; MCKEOWN, N. Reproducible network experiments using container-based emulation. In: ACM. **Proceedings of the 8th international conference on Emerging networking experiments and technologies**. [S.l.], 2012. p. 253–264.

HUAGMACHI. **Huagmachi/exp_EFattree is an experiment to compare the performance of EFattree with ECMP, PureSDN and Hedera.** 2018. <https://github.com/Huangmachi/exp_EFattree>. Acesso em: 20 ago. 2018.

IBM. **CPLEX Optimizer | IBM**. 2018. <https://www.ibm.com/analytics/cplex-optimizer>. Acesso em: 12 mar. 2018.

QIAN, Z.; HU, B.; YEUNG, K. L. An efficient routing algorithm in fat-tree data center networks. In: IEEE. **Global Communications Conference (GLOBECOM), 2016 IEEE**. [S.l.], 2016. p. 1–6.

RAGHAVAN, P.; TOMPSON, C. D. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. **Combinatorica**, Springer, v. 7, n. 4, p. 365–374, 1987.

SANTOS, M.; AGOULMINE, N.; RACHKIDY, E.; FERNANDES, S. Revisitando o problema de alocação de controladores sdn: Uma análise sobre o impacto do custo de recobrimento da rede. In: **Simpósio Brasileiro de Redes de Computadores (SBRC)**. [S.l.: s.n.], 2018. v. 36.

WANG, T.; SU, Z.; XIA, Y.; HAMDI, M. Rethinking the data center networking: Architecture, network protocols, and resource sharing. **IEEE access**, IEEE, v. 2, p. 1481–1496, 2014.

WILLIAMSON, D. P.; SHMOYS, D. B. **The design of approximation algorithms**. [S.l.]: Cambridge university press, 2011.

ZHANG, Y.; CUI, L.; ZHANG, Y. A stable matching based elephant flow scheduling algorithm in data center networks. **Computer Networks**, Elsevier, v. 120, p. 186–197, 2017.