FlatNet: Uma Rede Auto-Reconfigurável Distribuída e Concorrente com Topologia em Árvore Binária

Otávio A. O. Souza[1], **Caio Caldeira**[1], **Olga N. Goussevskaia**[1]

[1]Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG)
31270-901 - Belo Horizonte - MG - Brasil

{oaugusto,caio.caldeira,olga}@dcc.ufmg.br

Abstract. *In this paper we present FlatNet, a new binary-tree topology-restructuring protocol. FlatNet incrementally adapts the network topology in a decentralized and concurrent manner in response to the demand of the application it serves, bringing more frequently communicating nodes closer together through low-cost local restructurings. Unlike previous solutions, where binary search tree properties have to be maintained or only one communication request can be fulfilled at a time in the network, limiting overall performance, FlatNet does not depend on binary search tree properties and allows operations to occur concurrently. We prove that FlatNet is correct, provide bounds on worst-case performance, and present simulation results on request traces collected from real-world data centers.*

Resumo. *Neste trabalho apresentamos FlatNet, um novo protocolo de reestruturação de topologias de árvore binária. FlatNet incrementalmente adapta a topologia da rede de forma descentralizada e concorrente em resposta à demanda da aplicação que serve, aproximando os nós que se comunicam com maior frequência atráves de reestruturações locais de rede. Diferentemente de soluções existentes, que são baseadas em propriedades de pesquisa binária ou não permitem que operações ocorram concorrentemente na rede, limitando o desempenho, FlatNet é uma solução totalmente distribuída e concorrente que não necessita manter propriedades de árvore binária de busca. Nós provamos que FlatNet é correto, provemos limites de custo no pior caso e apresentamos resultados de simulações em sequências de requisições coletadas de centros de processamento de dados reais.*

1. Introdução

As redes de comunicação estão se tornado cada vez mais flexíveis: em termos de roteamento (redes definidas por *software* [Kreutz et al. 2015]), virtualização de infra-estrutura [Chowdhury and Boutaba 2010], e em termos de topologia física (tecnologias reconfiguráveis [Ghobadi et al. 2016, Liu et al. 2014, Farrington et al. 2010, Hamedazimi et al. 2014, Zhou et al. 2012]). A possibilidade de reconfiguração dessas redes, tanto pela migração de elementos terminais ou por alteração das conexões, permite torná-las reativas à demanda de comunicação: redes que se adaptam ao padrão de comunicação que elas servem de forma *online*.

Isso pode ser observado ao analisarmos que em uma rede auto-adaptativa, pares de nós que se comunicam com maior frequência podem ser aproximados topologicamente , economizando recursos, como a largura de banda e energia, e melhorando o desempenho de aplicações, como latência e vazão.

A demanda de comunicação presente em certas aplicações de redes, como em data centers, tende a não ser completamente aleatória e apresentar alguma estrutura: as matrizes de comunicação são geralmente esparsas e concentradas em pequenos grupos [Roy et al. 2015]. Em [Ghobadi et al. 2016] por exemplo, é mostrado que uma alta porcentagem dos pares de *rack's* em data centers não se comunicam entre si, de modo que 80% da comunicação total entre eles vem de apenas 1% dos pares. Em muitas dessas aplicações, o padrão de comunicação é conhecido por apresentar alguma localidade, ou seja, nós que se comunicaram no passado são prováveis de comunicarem novamente no futuro. Entretanto, estudos empíricos têm mostrado que, ainda assim, há dificuldade em estimar e prever a matriz de comunicação [Cao et al. 2016, Hu et al. 2015, Kandula et al. 2009], o que torna inviáveis soluções ótimas estáticas de configuração de topologias. Aplicações com essas características podem tirar grande proveito de redes que se reconfiguram dinamicamente de forma a otimizar sobre a demanda de comunicação.

No entanto, o projeto de redes auto-adaptativas pode conter vários desafios, já que a demanda de comunicação não é conhecida a priori, o que faz com que estes algoritmos tenham de reagir às requisições de comunicação de forma inteligente. Além disso, é importante que seja levado em consideração o custo-benefício de transformações na rede, visto que, embora ajustes mais frequentes possam melhorar o tempo de comunicação e sua viabilidade, podem também aumentar o custo do algoritmo. Entretanto, como mostrado em [Reiter et al. 2008, Schmid et al. 2016, Peres et al. 2019], mesmo com todas as dificuldades e a pouca informação a cerca desses algoritmos, eles são competitivos até mesmo com algoritmos ótimos *offline*, que conhecem a demanda de comunicação a priori.

A visão de redes auto-ajustáveis pode ser analisada como uma generalização do conceito de estruturas de dados de pesquisa em memórias auto-ajustáveis como *splay trees* [Sleator and Tarjan 1985], em que a estrutura é otimizada em razão dos acessos. Em particular, *splay trees* auto-ajusta com o padrão de acesso, aproximando nós mais requisitados da raiz: o trabalho de mover os nós é compensado pela redução do tempo de acesso para buscas futuras. Em [Sleator and Tarjan 1985] é mostrado que em termos de custo amortizado, *splay trees* tem um desempenho tão bom quanto qualquer árvore balanceada e estática ótima. Enquanto em estrutura de dados as requisições sempre se originam da raiz, em redes auto-adaptativas as requisições ocorrem entre pares arbitrários de nós, como é ilustrado na Figura 1 que mostra a diferença entre estrutura de dados e redes de comunicação.

Uma característica fundamental das estruturas de dados de pesquisa em memória são as propriedades de busca, ou seja, os valores das chaves são organizados de forma a manter a ordem relativa entre as subárvores a esquerda e a direita de cada nó. No entanto, em redes de comunicação, essa característica não é sempre necessária ou desejável. Redes *overlay*, por exemplo, são extremamente dinâmicas o que torna o custo de manutenção de propriedades de busca binária caras. Além disso, a restrição de manter a propriedades de busca binária limitam o número de possíveis topologias da rede, aumentando assim

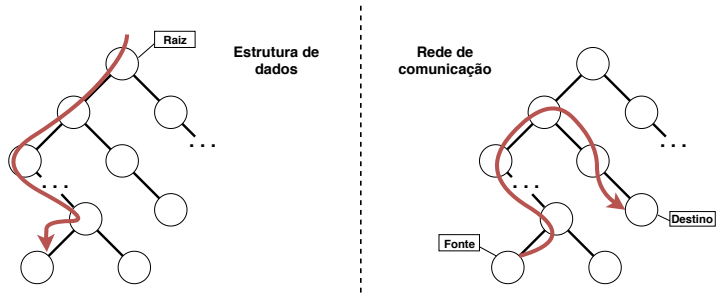


Figura 1. Diferença entre estruturas de dados e rede de comunição em árvore

as distâncias relativas entre os nós que se comunicam e, consequentemente, o custo de comunicação.

Existem poucas implementações de redes auto-ajustáveis que não mantêm propriedades de busca binária na rede. Um exemplo é o protocolo proposto em [Reiter et al. 2008], referido aqui como *Flattening*, que é um algoritmo auto-adaptativo para redes binárias que utiliza de heurísticas semelhantes às da *splay tree*. *Flattening* não exige que a rede de comunicação mantenha uma estrutura binária de busca, característica interessante para redes par-a-par que, por se tratarem de aplicações de caráter dinâmico e transitório (nós aderem e deixam a rede a todo instante), fazem com que seja difícil manter a propriedade de ordem dos elementos. No entanto, *Flattening* só permite uma execução sequencial de requisições de comunicação, o que pode ser lido como uma das principais limitações do algoritmo.

O presente trabalho propõe um sistema completamente distribuído e concorrente, denominado *FlatNet*, que estende e generaliza as ideias por trás do protocolo *Flattening* [Reiter et al. 2008]. Além disso, a concorrência em operações independentes na rede é garantida através da utilização de algumas técnicas de sistemas distribuídos, desenvolvidas em *DiSplayNet* [Peres et al. 2019]. A avaliação do desempenho de *FlatNet* é realizada através de duas métricas, sendo elas: (1) O custo amortizado de reconfigurações; (2) *makespan*, que corresponde à quantidade de tempo necessária para completar uma sequência de requisições. Além disso, é mostrado que, assim como *DiSplayNet*, nosso algoritmo é livre de *deadlocks* e inanição. São também relatados resultados de simulações que esclarecem as oportunidades e limitações de se aproveitar da localidade, bem como da concorrência em *FlatNet*. *FlatNet* foi testado através de dados reais coletados de centros de processamento executando operações de *MapReduce*, banco de dados e armazenamento.

2. Trabalhos Relacionados

Soluções de redes reconfiguráveis nas quais a topologia é otimizada sobre a demanda de comunicação estão ganhando notoriedade principalmente em cenários de data centers [Ghobadi et al. 2016]. Algumas soluções utilizam-se de estimativas ou *snapshots* da demanda para otimizar a topologia periodicamente (através de programação linear ou heurísticas [Singla et al. 2010, Ghobadi et al. 2016]). Entretanto, essas soluções não levam em consideração o custo-benefício de reconfigurações, o que, em termos, define a viabilidade deste tipo de rede.

A primeira proposta de redes auto-adaptativas como uma generalização de *splay trees* foi apresentada em [Reiter et al. 2008]. Nesta protocolo, quando uma mensagem é

gerada, uma série de transformações locais ocorrem entre os nós-fonte e destino no intuito de torná-los próximos na rede. Enquanto o protocolo proposto vale-se das mesmas propriedades de análise amortizada presentes em *splay trees*, o mesmo não suporta concorrência entre operações. Ou seja, apenas uma requisição é atendida por vez na rede, indo contrário as características dessas redes de comunicação e limitando o desempenho das aplicações às quais servem.

De forma independente, em [Schmid et al. 2016] é proposto um algoritmo de reestruturação de topologia de árvore binária de busca, denominado *SplayNet*. Este trabalho investiga também o custo-benefício de algoritmos adaptativos do ponto de vista teórico: o algoritmo proposto realiza, assim como *splay trees*, uma série de rotações entre os nós fonte e destino até atingir o ancestral comum de ambos os nós na árvore, então, completado a série de rotações, os nós tornam-se vizinhos e uma comunicação pode ser estabelecida. No entanto, assim como *flattening*, esse algoritmo não permite concorrência entre operações.

Para contornar esse problema, em [Peres et al. 2019] é proposto *DiSplayNet*, uma primeira abordagem totalmente distribuída que permite concorrência entre operações. *DiSplayNet* utiliza-se de diretivas de concorrência para permitir que requisições independentes possam ser atendidas ao mesmo tempo. São revistas também possíveis complicações que surgem devido ao conflito de operações, como custo adicional, (*overhead*). Entretanto, até certo ponto, isso não é um limitador para a viabilidade do algoritmo, que muita das vezes apresenta resultados ainda melhores do que os da versão sequencial.

Este trabalho difere-se de DiSplayNet em dois pontos: em primeiro lugar, quanto à forma como as mensagens são enviadas. Em DiSplayNet ambos os nós, fonte e destino sofrem operações de splays até tornarem-se vizinhos para então iniciar a comunicação. Já em FlatNet, a mensagem é enviada do nó fonte e encaminhada por nós intermediários até atingir o destino realizando reconfigurações no caminho. Em segundo lugar, este algoritmo emprega operações de semi-*splays* ao contrário de operações normais de *splays*. Para facilitar a visão da contribuição trazida pela pesquisa, a Tabela 1 mostra a classificação dos algoritmos quanto à concorrência e estrutura mantida em árvore. Assim, é possível dizer que este trabalho preenche uma lacuna no nicho de algoritmos auto-adaptativos ao estender *flattening* sobre concorrência em árvores binárias.

	Árvore binária	**Árvore binária de busca**
Sequencial	**Flattening**	**SplayNet**
Concorrente	**FlatNet**	**DiSplayNet**

Tabela 1. Algoritmos auto-adaptativos: classificação dos algoritmos quanto à concorrência e estrutura mantida em árvore.

3. Modelagem

A rede aqui analisada é constituída de um conjunto $V = \{v_1, v_2, v_3..., v_n\}$ de n nós. A estrutura de comunicação é definida como um grafo $T = (V, E)$ conectando cada nó através de uma topologia de árvore binária. A comunicação entre nós vizinhos se dá pela passagem de mensagem seguindo um modelo síncrono de comunicação. Para que quaisquer

dois nós não vizinhos se comuniquem, as mensagens são roteadas por nós intermediários no caminho. A entrada para a rede será uma demanda de comunicação, dada por uma sequência $\sigma = (\sigma_1, \sigma_2, \sigma_3..., \sigma_m)$ de m requisições de comunicação da forma $\sigma_i = (s, d)$, sendo s a fonte e d o destino de comunicação. Este algoritmo considera todos os nós como já estando inicialmente dispostos na rede, inicializados com informação de seus vizinhos imediatos, e com uma tabela de roteamento na árvore.

Neste modelo, cada nó envia uma mensagem por vez na rede, e somente após receber confirmação de envio outra mensagem pode ser enviada. Assim como em *flattening*, as confirmações são enviadas de forma direta por mecanismos fora da rede. Mensagens que não estão em transmissão, são armazenadas em ordem de chegada por meio de buffers locais nos nós. Quando uma mensagem é retirada do buffer para ser enviada, o tempo corrente é atribuído à mensagem e usado como valor de prioridade entre operações.

Para que a concorrência possa ser estudada, o tempo do modelo é dividido em *rounds*: em cada *round*, múltiplos (independentes) nós podem realizar reconfigurações locais concorrentemente, visando minimizar tanto o trabalho demandado (número de reconfigurações) quanto o tempo necessário para completar toda sequência σ de requisições. Já o custo do algoritmo é analisado e definido em termos de trabalho e tempo:

Definição 1 ***(Custo de trabalho)****: Considerando uma árvore inicial T_0 e uma sequência de requisições de comunicação, o custo total de trabalho pode ser definido como o número de reconfigurações locais e de operações de encaminhamentos necessários para enviar uma sequência σ de mensagens.*

Em termos de concorrência, o objetivo é minimizar o tempo total demandado para executar m operações.

Definição 2 ***(Custo de tempo)****: Considerando uma árvore inicial T_0 e uma sequência de requisições $\sigma = (\sigma_1, \sigma_2, \sigma_3..., \sigma_m)$, é definido como makespan o tempo decorrido entre o início da primeira requisição σ_1 e o término da última requisição σ_m.*

É importante que seja também apresentada a definição de menor ancestral comum estabelecida para as análises de funcionamento do algoritmo:

Definição 3 ***(Menor Ancestral Comum)****: O menor ancestral comum (MAC) entre dois nós u e v corresponde ao nó mais próximo de u e v que contenha ambos como descendentes. Note que um nó é menor ancestral comum dele mesmo com qualquer outro nó descendente a ele.*

4. FlatNet

Em alto nível, FlatNet funciona da seguinte forma: quando uma mensagem é enviada de um nó s para um dado nó d, o algoritmo executa uma série de reconfigurações locais da topologia da rede, enquanto a mensagem vai sendo transmitida por nós intermediários. FlatNet gradativamente modifica a topologia de forma a tornar próximos nós que se comunicam com maior frequência.

Para realizar as reconfiguração de forma distribuída, FlatNet depende dos seguintes conceitos:

1. *Tabela de roteamento:* Cada nó possui uma tabela de roteamente para todos os demais nós na rede. Essa tabela é responsável por direcionar as mensagem e operações de reconfiguração.
2. *Reconfigurações locais:* Para modificar a topologia da rede, FlatNet utiliza-se de reconfigurações locais de **semi-*flatting*** [Reiter et al. 2008], essas operações alteram de conexões em quantidade constante nós. Como as operações são locais, a tabela de roteamento é atualizada durante as rotações.
3. *Formação de clusters:* Para certificar a corretude das transformações, bem como para evitar *deadlocks*, cada operação de semi-*flattening* requer a formação de um *cluster*(conjunto de nós que terão conexões modificadas). Em cada *cluster* somente uma operação de semi-*flattening* pode ocorrer por *round*.

4.1. Tabela de Roteamento

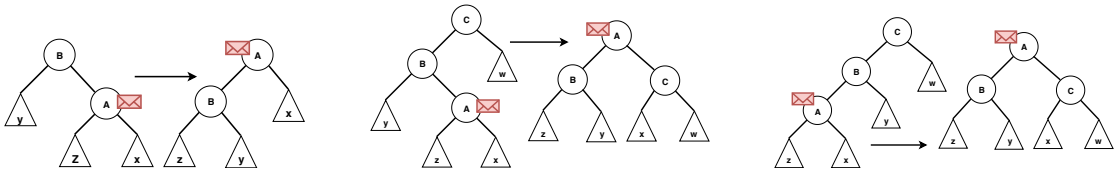
O algoritmo de *FlatNet* não faz qualquer suposição sobre a implementação da tabela de roteamento, apenas que ela deve informar acerca do roteamento da mensagem, para o nó pai, filho da direita e filho da esquerda. No caso, se o destino não pertence aos descendentes do nó, a mensagem deve ser roteada para cima na rede, caso contrário, a tabela indica um dos filhos que a mensagem deve seguir. Entretanto, a tabela de roteamento não é o único mecanismo que pode ser utilizado. O algoritmo de *flattening*, por exemplo, utiliza de *flooding* para enviar mensagens. Neste trabalho utilizamos da tabela por simplicidade de implementação.

4.2. Reconfigurações Locais

Em particular, FlatNet emprega dois tipos de operações de reconfiguração enquanto a mensagem percorre o caminho entre o nó fonte e o destino. Uma operação de *bottom-up* é utilizada quando a mensagem se move para cima na árvore essa operação termina quando o destino ou o menor ancestral comum (MAC) for alcançado. Caso o nó fonte já corresponde ao nó mais alto, não é realizada a operação de *bottom-up*. Caso a mensagem não tenha encontrado o destino, operações de *top-down* são empregadas para mover a mensagem para baixo na árvore até que atinja o destinatário. Ambos operações empregadas no algoritmo foram desenvolvidos em Flattening [Reiter et al. 2008]. A seguir descrevemos as operações detalhadamente.

4.2.1. *Bottom-up Flattening*

A operação de *bottom-up* de FlatNet é empregada quando a mensagem está navegando para cima na rede, do nó fonte para o nó destino, e começa do nó fonte, indo em direção ao MAC de ambos os nós fonte e destino. Se o nó fonte já corresponde ao MAC, nenhuma operação é executada. Os três tipos de operações são descritos na figura 2. Perceba que o *bottom-up zig* 2(a) e o semi *zig-zig* 2(b) consistem em somente uma única rotação, enquanto semi *zig-zag* 2(c) realiza duas rotações, e que a operação de *zig* somente é empregada quando o nó destino ou o menor ancestral comum se encontra no pai do nó que a mensagem pertence.

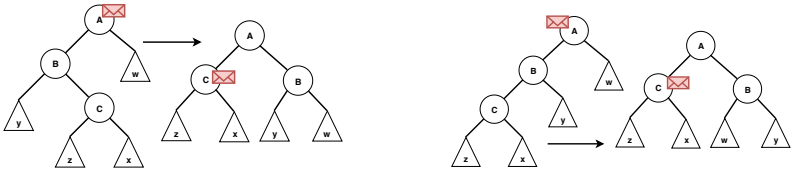


(a) zig (b) semi zig-zag (c) semi zig-zig

Figura 2. Operações bottom-up: São aplicadas conforme a estrutura do caminho da mensagem em direção ao MCA. Note que a operação de *zig* e semi *zig-zig* consistem em somente uma única rotação, enquanto semi *zig-zag* realiza duas rotações.

4.2.2. *Top-Down Semi-Flattening*

Esse segundo esquema é usado quando a mensagem está navegando para baixo na rede. Esta operação começa a partir do MAC e termina quando o nó destino é encontrado. A Figura 3 ilustra as operações de *top-down*. É necessário notar que a mensagem sempre termina em um nó inferior ao inicial. Esta operação termina quando o nó destino é alcançado pela mensagem.



(a) semi zig-zag (b) semi zig-zig

Figura 3. Operações Top-down: as operações são empregadas quando a mensagem trafega para baixo na rede em direção ao destino.

4.3. *Clusters*

Como as reconfigurações realizadas por um nó portando uma mensagem precisa de acesso exclusivo aos nós vizinhos para alterar as conexões, é utilizado o conceito de *clusters* definido em [Peres et al. 2019] para garantir a consistência da rede. Um *cluster* é formado por todos os nós que terão conexões alteradas durante uma operação e que concordam sobre qual nó tem preferência de operação. Somente um dos nós pertencente ao *cluster* pode realizar operação por *round*. Quando um nó está portando mensagem, uma operação de formação de *cluster* é inicializada antes de executar a operação em questão. Após a formação do *cluster* as conexões podem ser alteradas simultaneamente.

4.4. Algoritmo FlatNet

Com esses conceitos em mente podemos agora apresentar o algoritmo de Flatnet. As mensagens realizam progresso em direção ao destino através de operações de reconfigurações locais na rede, denominadas de *Steps*. Quando um nó possui uma mensagem que deve ser enviada, primeiramente, é consultado a posição relativa de seus vizinhos no caminho para o destino a fim de decidir qual tipo de operação aplicar. Uma vez que decidido qual operação, o nó realiza a formação de *cluster* como desenvolvido em DiSplayNet. Caso

ocorra conflito na formação do *cluster*, o tempo de entrada na rede da mensagem é utilizado como prioridade. Com o *cluster* formado a opereção pode ser executada na rede. Após a operação ser aplicada, a mensagem reduz a distância ao nó destino, que dependendo do tipo de operação pode permanecer ou migrar para outro nó intermediário. A invariante de *FlatNet* descrito pelo algoritmo 1 reside exatamente nas operações: enquanto uma mensagem não alcança o nó destino, operações de reestruturações são realizadas pelo nós que transmitem a mensagem pelo caminho.

Algorithm 1 FlatNet

1: (* mediante requisição (u, v) em T *)
2: $w = MAC_T(u, v)$
3: $T' =$ **step** requisição de u para raiz de w (operações bottom-up)
4: $T'' =$ **step** requisiçao de w para v (operações top-down)
5: (*T'' corresponde ao estado atual da rede após completar requisição (u, v)*)

5. Análise

A corretude e análise do desempenho da FlatNet são apresentadas a partir dos seguintes critérios.

Deadlocks e inanição: Para permitir que FlatNet execute operações concorrentes e em tempo viável, são utilizadas técnicas apresentadas em [Peres et al. 2019]. No *Teorema 1* de *DiSplayNet* é mostrado que, seguindo o modelo de *clusters* e prioridades, todas as atualizações de conexões são consistentes e o algoritmo é livre de *deadlocks* e inanição. Isso advém do fato de que cada nó entra em consenso sobre somente um *cluster* em cada *round*, originado pelo nó de maior prioridade na vizinhança.

Trabalho e tempo em cenário sequencial: Uma vez que são utilizadas as mesmas operações de semi-*flattening*, a análise de trabalho e tempo para o algoritmo de *FlatNet* sequencial é a mesma análise presente em [Reiter et al. 2008]. Como no caso sequencial não há paralelismo entre operações, o custo de trabalho e de tempo são sempre equivalentes. Em *flattening* é feita uma análise amortizada do trabalho utilizando o método de potencial. Nesta análise é mostrado que para uma sequência de m requisições o trabalho total realizado é $O(m \log n)$.

Trabalho e tempo em cenário concorrente: A análise de *FlatNet* para o cenário concorrente é equivalente a análise de [Peres et al. 2019]. Nesta análise é considerado o custo adicional de trabalho gerado pela concorrência (*overhead*) entre operações. Esse custo adicional de trabalho se deve aos conflitos gerados por mensagens que possuem nós em comum sobre o mesmo *round*, resultando em pausas e *bypasses*. Neste cenário concorrente, o custo de tempo é sempre menor ou igual ao custo de trabalho. Em particular, o custo de trabalho total é dado por $O(nm \log n)$, n correspondendo ao *overhead*. E para o tempo, no pior caso, em que não existe paralelismo entre operações, o custo de tempo do algoritmo é igual ao custo de trabalho total.

6. Experimentos

Para para que a análise seja completa e o desempenho de *FlatNet* possa ser reportado, ambos em termos do custo de trabalho e tempo, foram conduzidas simulações do algoritmo com cargas de trabalho reais. Para realizar os experimentos, foi implementado o

algoritmo descrito nas seções anteriores, assim como em [Peres et al. 2019], no simulador Sinalgo[Group 2007]. Sinalgo é uma plataforma de simulação para validar algoritmos distribuídos. Para melhor avaliar o algoritmo em análise e os resultados das simulações, também foram implementados cinco *baselines*. Consideramos as seguintes implementações: **Árvore binária balanceada (bt)**: uma árvore binária com custo de comunicação $O(\log n)$, **Árvore binária estática ótima (opt):** árvore binária de busca ótima usando programação dinâmica proposto em [Schmid et al. 2016], **SplayNet (sn)**, **DiSplayNet (dsn)**, **Flattening (flng)**. É importante perceber que o algoritmo ótimo estático possui conhecimento a priori da distribuição de comunicação e que assim como a árvore binária balanceada não tem custo de reconfiguração, apenas de roteamento.

6.1. Configuração dos experimentos

A carga de trabalho dos experimentos foi gerada amostrando $m = 10000$ requisições seguindo as distribuições de probabilidade descritas a seguir, e para cada configuração de experimento foram executadas 30 sequências de requisições diferentes a fim de obter a média e intervalo de confiança.

Além dos datasets reais, primeiramente, testamos *FlatNet* sobre uma sequência de requisições artificial, sem qualquer estrutura de localidade temporal ou espacial. Para gerar essa sequência de requisições, cada nó escolhe, de forma uniforme, outro nó na rede para enviar uma mensagem. A matriz de comunicação para essa distribuição pode ser visualizada na Figura 4. O objetivo deste dataset é mostrar o desempenho do algoritmo comparado com os demais *baselines* para o pior cenário, no qual não existe qualquer estrutura de localidade. A melhor configuração para esta sequência corresponderia a árvore binária balanceada.

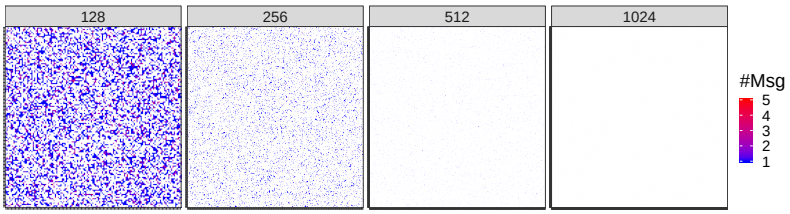
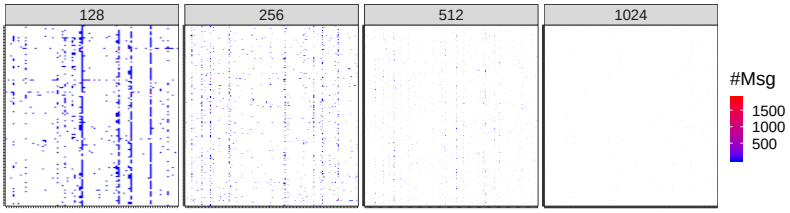
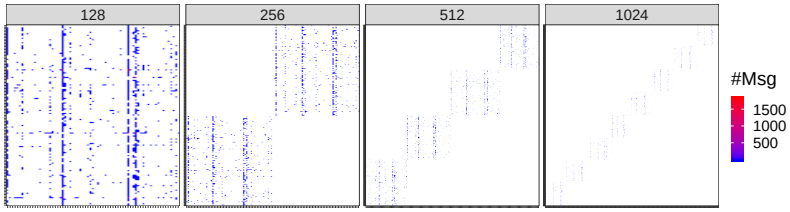


Figura 4. Matriz de comunicação: Note que a distribuição de requisições é completamente uniforme sobre os pares para este dataset.

Para gerar a sequência de requisições reais, foi levantado um dataset coletado e publicados por [Ghobadi et al. 2016]. O dataset descreve a função de densidade de probabilidade sobre 8367 pares de comunicação em uma rede constituída de 128 nós. Os 128 nós foram selecionados de dois grupos de produção (executando trabalhos de MapReduce, banco de dados e sistema de armazenamento). No sentido de estudar o desempenho de FlatNet sobre escalas maiores de nós, foi replicado o padrão de comunicação para redes maiores. Em particular, o enfoque foi em redes de tamanho 128 e 1024. Para avaliar o desempenho sobre estruturas de comunicação que podem se formar na rede, foi simulada uma formação de grupos de nós que não trocam mensagens entre si. As matrizes de comunicação mostradas nas Figuras 6.1 e 6.1 a seguir representam as sequências geradas sem e com formação de clusters respectivamente. Para estes cenários, é esperado que o algoritmo ótimo estático desempenhe o melhor trabalho, já que conhece a distribuição à priori.



(a) Dataset 1



(b) Dataset 2

Figura 5. Matrizes de comunicação: Há uma concentração de comunicação entre alguns paras de nós e para o segundo dataset podemos ver que apenas grupos bem definidos de nós se comunicam.

7. Resultados

Os resultados das simulações foram avaliados em termos do trabalho de reconfigurações, do tempo gasto para completar uma sequência de requisições e o número de operações concorrentes em cada algoritmo. A partir dos dados obtidos é possível observar os benefícios do *FlatNet* nos cenários dos experimentos, como pode ser visto nas observações a seguir.

Distribuição Uniforme: Neste experimento foi usada uma sequência de requisições sem qualquer tipo de localidade. E, como esperado, a configuração de árvore binária balanceada e o algoritmo ótimo estático apresentaram resultados equivalentes sobre o trabalho, além dos melhores resultados dentre os algoritmos. Na Figura 6 é possível observar o trabalho total em número de reconfigurações e roteamento e a função de distribuição cumulativa para o trabalho em número de reconfigurações. Devido à natureza descentralizada de *DiSplayNet* e *FlatNet* também é possível observar um custo adicional comparado com as versões sequenciais. Esse custo está presente na análise de pior caso desses algoritmos (Sec 5).

Ao mesmo tempo, em termos de *makespan*, podemos observar os benefícios da concorrência mostrados nos gráficos da Figura 7. Os algoritmos *DiSplayNet* e *FlatNet* apresentaram maior vazão e menor *makespan* mesmo nesse tipo de experimento. *FlatNet* obteve resultados ainda melhores que *DiSplayNet* o que atribuímos à falta de ordem de busca mantida pelos nós na rede. Por último, foi analisado o número de *clusters* por *round* para os algoritmos *FlatNet* e *DiSplayNet*. Como mostrado na Figura 8, o algoritmo de *FlatNet* possui um paralelismo maior comparado a *DiSplayNet*.

ProjecToR Dataset 1: Neste cenário os algoritmos foram testados sobre sequência de requisições com presença de localidade. Os gráficos da Figura 9 mostram o custo de trabalho das reconfigurações e a função de distribuição cumulativa para o trabalho. Neste experimento, os algoritmos adaptativos apresentaram grande melhora em termos de tra-

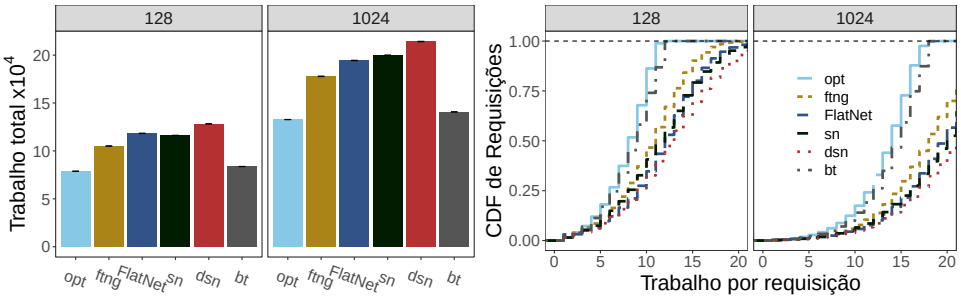


Figura 6. Distribuição uniforme: Trabalho total e CDF de requisições

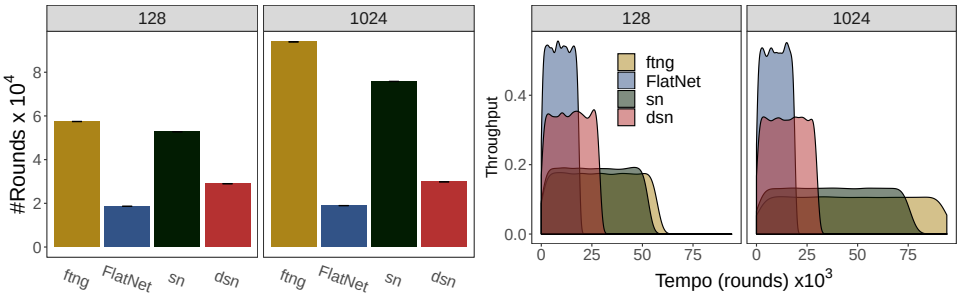


Figura 7. Distribuição uniforme: Makespan e vazão

balho, alcançando valores próximos do ótimo estático. Nos gráficos ainda é possível ver que o custo adicional dado pelas implementações concorrentes é negligenciável perto das versões sequenciais, inclusive podendo ser até melhores em alguns experimentos em presença de localidade. Assim como no cenário anterior, *FlatNet* e *DiSplayNet* apresentaram resultados muito superiores às versões de algoritmos em árvore binária de busca.

Em termos de vazão e *makespan* é nítido a vantagem das implementações concorrentes, com *FlatNet* alcançando os melhores resultados como mostrado na Figura 10.

ProjecToR Dataset 2: Por fim, foi testado os cenários com localidade e presença de estruturas de comunicação, nos quais existem grupos independentes de nós que não se comunicam entre si. Como esperado, os algoritmos concorrentes têm maior desempenho com essa configuração. A Figura 12 mostra o trabalho de reconfiguração e a função de distribuição cumulativa para o trabalho. Os algoritmos auto-adaptativos apresentaram resultados bem melhores em relação à árvore binária balanceada. Além disso, é possível ver que devido à estrutura de comunicação o tamanho da rede não acarreta em maior custo de trabalho.

Na Figura 13, é possível observar que o *makespan* e a vazão de *FlatNet* alcançam os melhores resultados neste experimento. O gráfico da Figura 14 também mostra um paralelismo maior devido a formação de grupos de comunicação.

8. Conclusão

Em suma, é possível considerar que o presente trabalho preenche uma lacuna no nicho de algoritmos auto-adaptativos e distribuídos. Neste artigo, foi explorado de forma nova e mais extensa o conceito de [Reiter et al. 2008] para um protocolo totalmente distribuído que dinamicamente reestrutura a rede de forma a reduzir a latência de

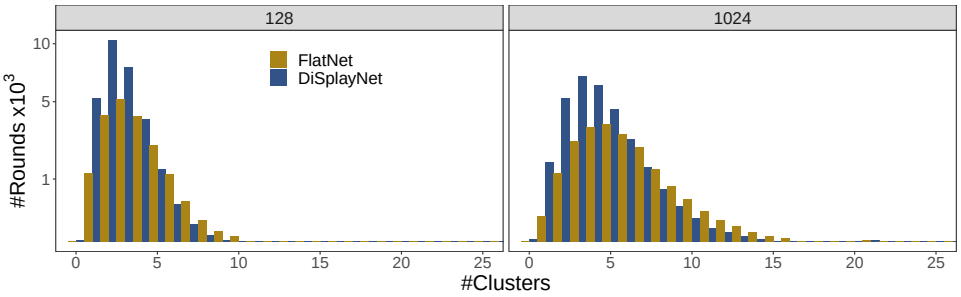


Figura 8. Número de clusteres concorrentes por round

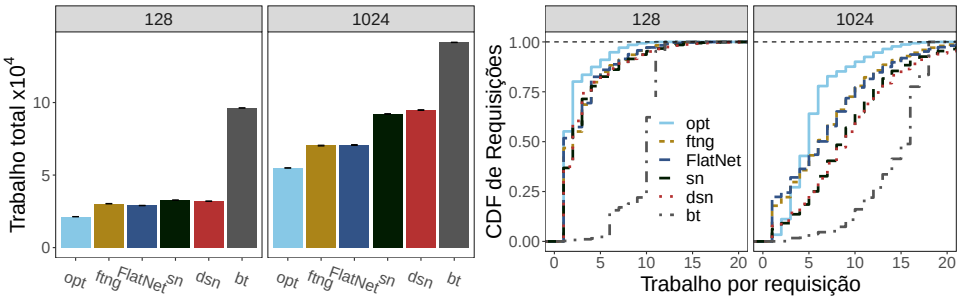


Figura 9. ProjecToR Dataset 1: Trabalho total e CDF de requisições

comunicação. Porém, ao contrário de *flattening*, *FlatNet* permite que múltiplas e independentes operações ocorram de forma concorrente. Desta forma, as aplicações apresentam um desempenho melhor em termos de *makespan*. Além disso, foi evidenciado o custo-benefício de permitir concorrência em detrimento de um custo adicional (overhead) de trabalho. Pôde ser observado também que em vários cenários esse custo não acarreta perda de desempenho e, curiosamente, muitas vezes reduz o trabalho total. A abordagem utilizada neste trabalho pode ser vista então como tendo sido capaz de confirmar os benefícios sobre diferentes padrões de comunicação coletados de data centers.

Referências

Cao, Z., Kodialam, M., and Lakshman, T. (2016). Joint static and dynamic traffic scheduling in data center networks. *Biological Cybernetics*, 24(3):1908–1918.

Chowdhury, N. M. K. and Boutaba, R. (2010). A survey of network virtualization. *Computer Networks*, 54(5):862 – 876.

Farrington, N., Porter, G., Radhakrishnan, S., Bazzaz, H. H., Subramanya, V., Fainman, Y., Papen, G., and Vahdat, A. (2010). Helios: a hybrid electrical/optical switch architecture for modular data centers. *ACM SIGCOMM Computer Communication Review*, 40(4):339–350.

Ghobadi, M., Mahajan, R., Phanishayee, A., Devanur, N., Kulkarni, J., Ranade, G., Blanche, P.-A., Rastegarfar, H., Glick, M., and Kilper, D. (2016). Projector: Agile reconfigurable data center interconnect. In *Proceedings of the 2016 ACM SIGCOMM*, SIGCOMM '16, pages 216–229, New York, NY, USA. ACM.

Group, D. C. (2007). Sinalgo - simulator for network algorithms. `http://disco.ethz.ch/projects/sinalgo/index.html`. Accessed 10-May-2017.

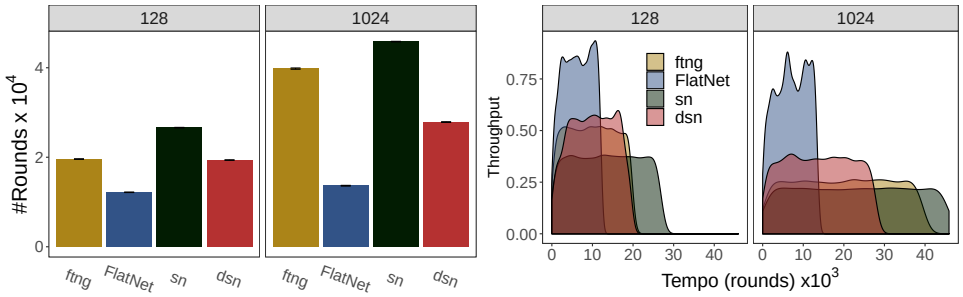


Figura 10. ProjecToR Dataset 1: Makespan e vazão

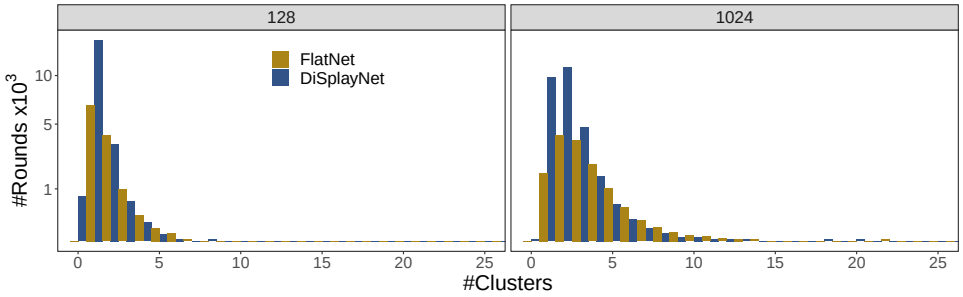


Figura 11. ProjecToR Dataset 1: Número de clusteres concorrentes por round

Hamedazimi, N., Qazi, Z., Gupta, H., Sekar, V., Das, S. R., Longtin, J. P., Shah, H., and Tanwer, A. (2014). Firefly: A reconfigurable wireless data center fabric using free-space optics. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 319–330. ACM.

Hu, Z., Qiao, Y., and Luo, J. (2015). Coarse-grained traffic matrix estimation for data center networks. *Computer Communications*, 56:25–34.

Kandula, S., Sengupta, S., Greenberg, A., Patel, P., and Chaiken, R. (2009). The nature of data center traffic: measurements & analysis. In *Proc. 9th ACM Internet Measurement Conference (IMC)*, pages 202–208.

Kreutz, D., Ramos, F. M., Verissimo, 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.

Liu, H., Lu, F., Forencich, A., Kapoor, R., Tewari, M., Voelker, G. M., Papen, G., Snoeren, A. C., and Porter, G. (2014). Circuit switching under the radar with reactor. In *NSDI*, volume 14, pages 1–15.

Peres, B., Souza, O., Goussevskaia, O., Schmid, S., and Avin, C. (2019). Distributed self-adjusting tree networks. In *Proc. IEEE INFOCOM*.

Reiter, M. K., Samar, A., and Wang, C. (2008). Self-optimizing distributed trees. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–12.

Roy, A., Zeng, H., Bagga, J., Porter, G., and Snoeren, A. C. (2015). Inside the social network's (datacenter) network. In *ACM SIGCOMM Computer Communication Review*, pages 123–137. ACM.

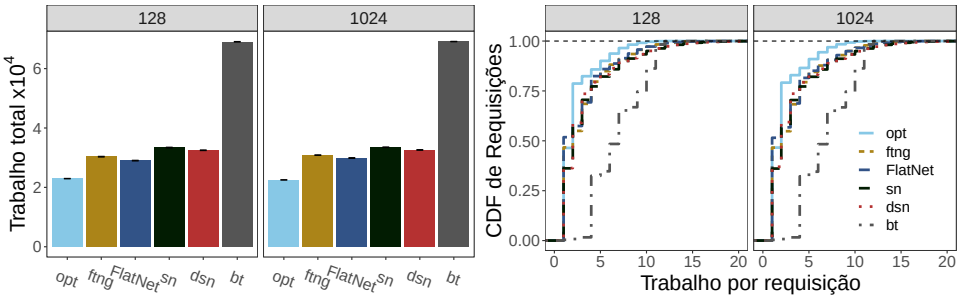


Figura 12. ProjecToR Dataset 2: Trabalho total e CDF de requisições

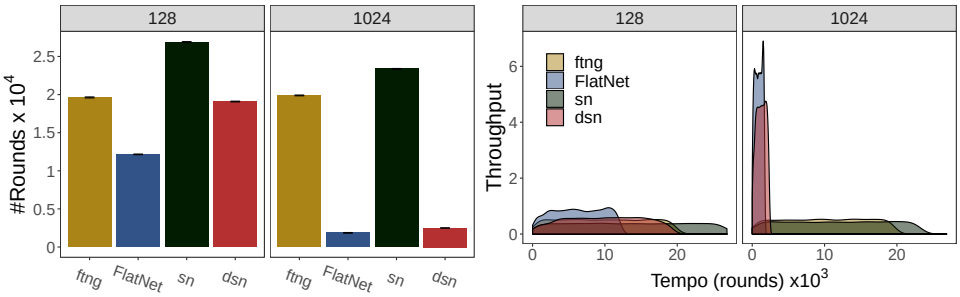


Figura 13. ProjecToR Dataset 2: Makespan e vazão

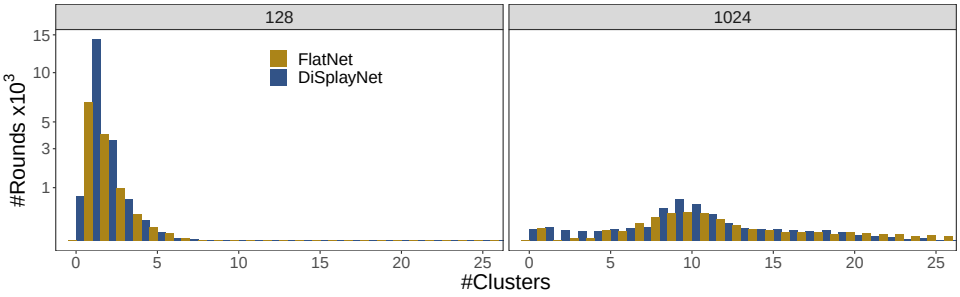


Figura 14. ProjecToR Dataset 2: Número de clusteres concorrentes por round

Schmid, S., Avin, C., Scheideler, C., Borokhovich, M., Haeupler, B., and Lotker, Z. (2016). Splaynet: Towards locally self-adjusting networks. *IEEE/ACM Trans. Netw.*, 24(3):1421–1433.

Singla, A., Singh, A., Ramachandran, K., Xu, L., and Zhang, Y. (2010). Proteus: a topology malleable data center network. In *Proc. ACM Workshop on Hot Topics in Networks (HotNets)*.

Sleator, D. and Tarjan, R. (1985). Self-adjusting binary search trees. *Journal of the ACM (JACM)*, 32(3):652–686.

Zhou, X., Zhang, Z., Zhu, Y., Li, Y., Kumar, S., Vahdat, A., Zhao, B. Y., and Zheng, H. (2012). Mirror mirror on the ceiling: Flexible wireless links for data centers. *ACM SIGCOMM Computer Communication Review*, 42(4):443–454.