Módulo para simulação de switches ópticos em redes auto configuráveis no simulador NS-3

Davi Lage Borges Souza
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais Belo Horizonte, Minas Gerais 30332–0250
Email: davi.lage@dcc.ufmg.br

***Abstract*—Os data centers desempenham um papel fundamental no suporte à crescente demanda por infraestrutura em nuvem e aplicações orientadas a dados. Entretanto, as topologias de rede tradicionais utilizadas nesses ambientes muitas vezes apresentam configurações fixas, incapazes de atender às exigências dinâmicas das redes, resultando em ineficiências operacionais. Para abordar esses problemas, este artigo introduz um novo módulo para o simulador NS-3, focado na avaliação e no aprimoramento do desempenho de topologias de rede reconfiguráveis em data centers. Por meio da implementação de topologias autoconfiguráveis, buscamos otimizar o uso de recursos e adaptar as redes a diferentes padrões de tráfego. O módulo permite o desenvolvimento e teste de algoritmos de reconfiguração, utilizando cenários de tráfego sintético e realista. Métricas como custo de roteamento, número de reconfigurações e utilização de switches são coletadas para analisar o desempenho dos algoritmos. Este trabalho promove o avanço da adaptabilidade e eficiência das redes em data centers, contribuindo para o desenvolvimento de infraestruturas mais inteligentes e autoadaptáveis.**

I. Introdução

Nos últimos anos, a relevância dos datacenters tem crescido exponencialmente, impulsionada pelo aumento de serviços que dependem de infraestrutura em nuvem e por aplicações cada vez mais orientadas a dados [1], [2]. Os datacenters desempenham um papel essencial no suporte a uma ampla variedade de aplicações, que vão desde plataformas de e-commerce até serviços de streaming em tempo real.

O avanço de aplicações distribuídas de aprendizado de máquina também tem sido um fator determinante para a expansão dos datacenters. Estudos recentes [3], [4] evidenciam como essas aplicações se beneficiam da infraestrutura robusta oferecida pelos datacenters para o processamento e análise de grandes volumes de dados. Entretanto, o crescimento da demanda traz consigo a necessidade de uma utilização mais eficiente dos recursos disponíveis [5], [6].

Para enfrentar esses desafios, diversas topologias de rede foram propostas com o objetivo de otimizar o uso de recursos e melhorar o desempenho das comunicações internas dos datacenters [7]–[9]. Contudo, as topologias convencionais possuem limitações, como sua estrutura fixa, que impede uma adaptação dinâmica às variações nas demandas da rede [10], [11].

Como alternativa, surgem as topologias autoconfiguráveis, projetadas para se adaptar em tempo real às mudanças no tráfego da rede [12], [13]. Essas soluções prometem maior eficiência no uso dos recursos e desempenho aprimorado da rede. No entanto, ainda há uma lacuna de ferramentas adequadas para o desenvolvimento e validação de algoritmos de reconfiguração [1]. Uma abordagem baseada em clusters, proposta em [14], explora a reconfiguração de switches ópticos para criar topologias dinâmicas e adaptáveis, permitindo o uso pleno de algoritmos autoconfiguráveis.

Nesse contexto, o simulador NS-3 apresenta-se como uma plataforma eficaz, oferecendo um ambiente flexível e escalável para a criação e avaliação de algoritmos de reconfiguração [15]. Para validar o módulo proposto, foram simulados cenários com tráfego sintético e realista, coletando métricas relevantes para medir o desempenho dos algoritmos. Este artigo apresenta uma abordagem pioneira para a validação de algoritmos de reconfiguração em datacenters, contribuindo para o aprimoramento da eficiência e adaptabilidade das infraestruturas em nuvem.

II. Referencial

A. Arquitetura da Rede

A rede é constituída por duas categorias de comutadores Top-of-Rack (ToR) switches e Inter/Intra cluster switches. Os ToRs se conectam a um nó host que gera o tráfego da rede, servindo como $source$ e $target$ das mensagens. Os switches inter-cluster conectam conjuntos de nós hosts e seus ToRs de forma a gerar grafos que permitam a comunicação entre quaisquer dois hosts na rede.

Para organizar os servidores, usamos a estrutura proposta em [14]. Isso se deve a capacidade de representar qualquer topologia com grau máximo k utilizando switches com p portas. A partir da separação dos servidores, agrupando-os sob um ToR_i de forma que podemos considerar os nós da camada de borda de uma rede com n nós apenas como cada um dos ToRs com índices $i = 0, 1, ..., n-1$, distribuídos em clusters $C_j = 0, 1, ..., c-1$, onde $c = \lceil \frac{n}{p} \rceil$ é o número de clusters calculado a partir do número de portas dos switches utilizados.

Os switches que compõem a spine são divididos em dois grupos conforme a função que realizam, sendo atribuídos a um Unit Cluster $SW^{\cup}$ caso interconectem ToRs de um mesmo conjunto ou a um Union Cluster $SW^{\times}$ caso conectem ToRs de conjuntos distintos como descrito na Fig. 1.

B. Estrutura SDN

Uma Rede Definida por Software (SDN) é um paradigma de construção de redes de computadores em que o processo de

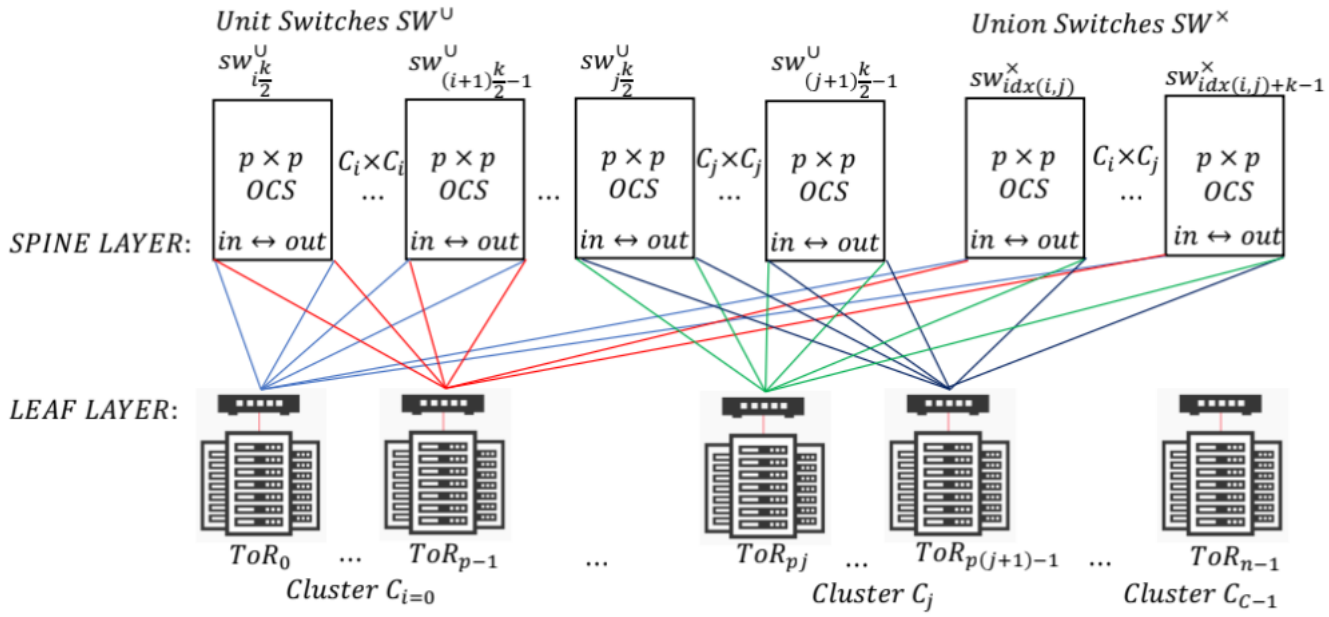

Fig. 1. (a) A rede OpticNet possui apenas uma camada de switches na spine e é separada em clusters. Cada conjunto de servidores conectados a um ToR pertence a um cluster de n ToRs com grau constante k

descoberta de rotas, encaminhamento de pacotes, resoluções de nomes são realizados utilizando um elemento chamado controlador, associado aos outros membros da rede (switches e servidores). [16], [17] O controlador é responsável por resolver conflitos em que não há uma decisão disponível para o membro da rede, e definir regras segundo as quais pacotes devem ser manipulados durante o tráfego.

Para definição das comunicações foi selecionado o protocolo OpenFlow [18] na versão 1.3. Esse protocolo define a comunicação entre os switches e um controlador por meio de um plano de controle (control plane) e permite a manipulação dos pacotes recebidos nas portas do switch utilizando um plano de dados (data plane). O protocolo implementa tabelas de fluxo (flow tables) que definem um pipeline para os pacotes recebidos nas portas do switch. Essa versão permite a utilização de tabelas múltiplas, agrupamento de portas, portas lógicas e a definição de medidores específicos para cada fluxo.

C. Redes SAN demand-oblivious e demand-aware

Uma categoria emergente de SDN são as Redes Auto Configuráveis (SAN) [1], em que o controlador define também um algoritmo segundo o qual alterações serão comunicadas aos membros da rede. O objetivo é otimizar a rede conforme uma expectativa de tráfego, ou sequência de mensagens, diminuindo o custo de comunicação entre nós que se comuniquem de forma mais frequente, gerando um custo amortizado o mais próximo do ótimo possível. As SAN podem ser categorizadas como independente da demanda (demand-oblivious) em que o controlador se comporta segundo um algoritmo que redefine a topologia da rede de uma forma predefinida [19]. Em contrapartida as redes cientes da demanda (demand-aware) implementam algoritmos em seus controladores de forma que as adaptações comunicadas aos membros da rede considerem os padrões de comunicação correntes entre os nós da rede. Assim, busca-se um efeito em que a topologia se torna mais adequada à sequência de mensagens esperada baseada na experiência passada. Isso se deve ao fato de que as comunicações em datacenter seguem matrizes desequilibradas, ou seja, a comunicação se concentra entre poucos pares de nós [20].

D. SplayNet

O algoritmo implementado no controlador dos switches da rede para validação do módulo é o SplayNet. Ele foi escolhido pela forma direta de definição das reconfigurações utilizando as rotações em sub-árvores por meio de rotações *zig-zig* e *zig-zag* em uma árvore binária de busca.

Conforme demonstrado em [13] o algoritmo aproxima a topologia da árvore ótima para uma sequência de mensagens, possuindo um custo amortizado limitado. Essa implementação é realizada utilizando um controlador capaz de armazenar as informações da estrutura abstrada da rede e realizar as rotações conforme necessário para uma dada mensagem σ_i.

E. Simulador ns-3

Considerando os requisitos para construção do ambiente virtual de simulação foi escolhido o NS-3 [15]. O NS-3 foi escolhido devido à sua qualidade no agendamento de eventos em uma série temporal utilizando ponto flutuante para mensurar o tempo em escala de nanosegundos, e capacidade de representar de forma detalhada as camadas de rede, características ausentes em simuladores como Mininet e OMNeT++.

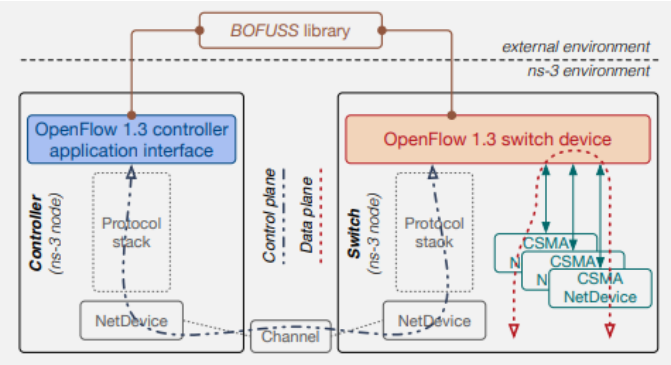

Fig. 2. Visão geral do módulo OFSwitch13 utilizado

De forma que a fidelidade para com o cenário em que os algoritmos de reconfiguração serão executados permita analisar o comportamento e encontrar potenciais falhas por limitações do ambiente prático o NS-3 se mostra como o mais adequado. Por se tratar de um simulador de eventos discretos toda a comunicação realizada durante a simulação leva em conta o agendamento dos eventos de trasmissão e recepção, permitindo a manipulação do fluxo do programa ao longo do tempo simulado. O simulador apresenta representações das camadas da pilha TCP por meio de classes especializadas. Para a camada física são definidas as larguras de banda e delay de propagação pelo canal, essa representação permite que os eventos de trasnsmissão e recepção de quadros disparem funções de callback. Para a camada de enlace, o simulador apresenta uma representação para o Network Interface Card (NIC) por meio da classe NetDevice, ao qual é atribuído um endereço MAC. Há duas implementações para redes cabeadas: Point-to-Point e CSMA, devido ao módulo openflow utilizado utilizar CSMA essa foi a forma utilizada no módulo do projeto. Há também representações da camada L3, utilizando o protocolo ARP, IPV4, e IPV6, assim, foi possível construir tabelas de roteamento e resolução de endereços utilizando pacotes com cabeçalhos conforme a RFC 826. Para a camada de transporte o NS-3 apresenta a possibilidade de construir soquetes para comunicação entre os nós, definindo portas e permitindo configurar pacotes a serem enviados entre os membros da rede para utilizar UDP ou TCP.

F. Módulo OfSwitch13

Para complementar o NS-3 com funcionalidades requeridas para simulação de SAN o módulo OFSwitch13 [21] foi instalado em conjunção ao simulador. Esse módulo permite a utilização do protocolo *OpenFlow 1.3*, fornecendo uma implementação base do switch com as capacidades de comunicação pelo plano de controle e plano de dados utilizando as tabelas de fluxo. A integração com a biblioteca BOFUSS [22], permite a utilização de comandos de configuração segundo a ferramenta *dpctl*, além do parsing de mensagens transitadas no plano de controle.

O controlador disponibilizado como base possui as capacidades básicas para controlar o fluxo de mensagens podendo armazenar os datapaths dos switches vinculados a ele. Além disso, o módulo apresenta uma solução em topologias baseadas em árvores binárias de busca para o problema que o simulador base apresenta que é a ausência de um mecanismo para resolução de ciclos na topologia da rede, apenas extender o comportamento da classe disponibilizada não é o suficiente para simular redes de datacenter com topologias distintas, requerendo uma adaptação do processo de montagem topológica descrito na subseção III-E.

G. Métricas (Reconfigurations Cost, Routing Cost)

Para avaliar o desempenho dos algoritmos de reconfiguração é necessário coletar dados da rede em tempo de funcionamento, sem interferir com a simulação do tráfego. Para isso, o simulador dispõe de mecanismos de tracing, que podem ser acoplados aos nós, como sondas, coletando informações ao longo da transmissão dos pacotes.

De forma a permitir um entendimento do desempenho do algoritmo foram selecionadas três métricas como definidas em [14], o *Makespan*, o *custo de reconfiguração* e o *custo de roteamento*. Considere uma sequência σ de m mensagens com origem s_i e destino d_i, uma rede inicial no estado $N_0 \in \mathcal{N}$ e um algoritmo de reconfiguração $\mathcal{A}$. Para uma mensagem $\sigma_i(s_i, d_i) \in \sigma$, denotamos por b_i o momento de envio da mensagem a partir do nó s, e por e_i o instante em que é entregue ao nó d. O cálculo do *Makespan* é dado por

$$Makespan(\mathcal{A}, N_0, \sigma) = max_{1 \leq i \leq m} e_i - b_1 \quad (1)$$

Considerando a sequência de mensagens, temos que o *custo de serviço* será o custo de rotear a mensagem ao longo do menor caminho $dist_{N_{e_i}}(s_i, d_i), N_{e_i} \in \mathcal{N}$, o comprimento do caminho $dist_{e_i}(s, d)$ na árvore resultante N_{e_i}, ou seja, após a entrega de σ_i, correspondendo ao número de encaminhamentos do pacote na rede, em que cada salto tem um custo de 1 unidade. O custo de serviço é dado por:

$$Srv(\mathcal{A}, N_0, \sigma) = \sum_{i=1}^{m} (dist_{N_{e_i}}(s_i, d_i) + 1) \quad (2)$$

Para avaliar a eficiência do algoritmo de reconfiguração em termos do custo gerado a partir das requisições de ajustes é usado o *custo de reconfiguração*. Seja ele definido como o número de ajustes realizados pelo algoritmo, cada um com custo α. Seja $\mathbb{I}_{S=S'}$, uma função indicadora que mostra se um set S é igual ao set S'. Assim, o custo de reconfiguração da rede é dado por:

$$Adj(\mathcal{A}, N_0, \sigma) = \alpha \sum_{i=1}^{t_{max}} \mathbb{I}_{M(x,t)=M(x,t-1)} \quad (3)$$

Por fim, sendo $t_{max} = \max_{1 \leq i \leq m} e_i$ o *custo total de trabalho* pode ser calculado como:

$$Cost(\mathcal{A}, N_0, \sigma) = Srv(\mathcal{A}, N_0, \sigma) + Adj(\mathcal{A}, N_0, \sigma) \quad (4)$$

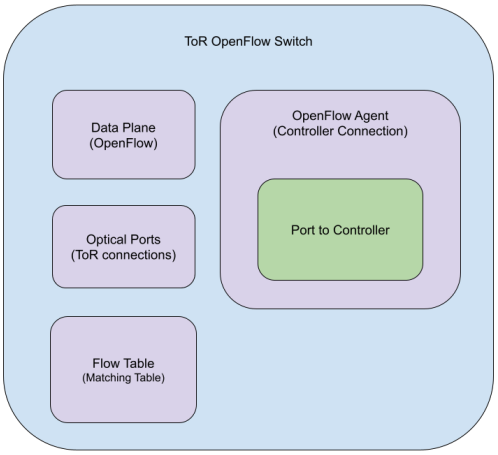

Fig. 3. Arquitetura da implementação do switch utilizando um controlador OpenFlow para construir a tabela de resolução de endereços. O comportamento é definido pelo controlador, sendo necessárias adaptações para o ToR e o switch intra/inter cluster.

III. Metodologia

A. Descrição do módulo (visão geral)

O módulo desenvolvido consiste em um template para leitura de um arquivo de entrada, geração da representação da estrutura de comunicação, sequenciamento temporal das mensagens, estabelecimento das rotas e uso de um controlador extensível para simulação de switches ToR e switches inter-cluster e intra-cluster.

B. ToR

O ToR switch utiliza um controlador especializado para definir o comportamento de um comutador de borda e agregação simultaneamente. Para lidar com mensagens vindas dos nós vinculados a ele o ToRs possui um fluxo que determina o encaminhamento de mensagens ARP para resolução utilizando o controlador geral da rede por meio de um switch intra-cluster. Utilizando mensagens codificadas utilizando o dpctl [22] o ToR se comunica com um controlador designado somente para ele, de forma que cada ToR toma decisões de forma distribuída.

Cada ToR apresenta uma tabela L2 própria mapeando as portas designadas para cada endereço MAC descoberto na rede, vinculando-os por meio do endereço IP. Além disso ao gerar a entrada na tabela CAM o controlador do switch gera uma mensagem criando uma nova entrada na tabela de fluxo designando o encaminhamento de quaisquer mensagens direcionadas para aquele mesmo IP e MAC, que utilize o mesmo protocolo, para a porta associada. A mensagem *dpctl* para essa nova entrada tem o formato

```
flow-mod cmd=add,table=0,prio=500
eth_type=0x0800,ip_proto=17,
ip_dst=dstIp,eth_dst=dstMac
apply:output=outPort;
```

Essa mensagem define que é uma modificação de fluxo (*flow-mod*) em que o comando é adicionar uma nova entrada (*cmd=add*) à tabela 0, com prioridade 500, em que pacotes do tipo IPv4 (*eth_type=0x0800*) com protocolo UDP (*ip_proto=17*) com destino ao endereço *dstIp*, e MAC *dstMac* sejam redirecionados para a porta de número *outPort* (*apply:output=outPort*). Ao instanciar a classe foi reservada a porta *1* para o nó conectado ao ToR, de forma que essa conexão funciona de forma particular, redirecionando os pacotes gerados pelo nó terminal para o controlador do ToR e encaminhando pacotes direcionados ao nó terminal de forma automática.

C. Classe do switch inter-cluster e intra-cluster

Para os switches inter-cluster e intra-cluster foi utilizado uma extensão do controlador base do módulo OFswitch13 [21]. Esses switches são conectados a um controlador único, de forma a representar uma estrutura de datacenter como a descrita em [5], [6], considerando a topologia construída como sendo o domínio desse controlador. Cada switch recebe uma instrução básica de sempre recorrer ao controlador para tomar decisões de resposta ou redirecionamento de pacotes. Para isso é adicionada a entrada

```
flow-mod cmd=add,table=0,prio=maxPrio
apply:output=ctrl:128
```

Essa entrada modifica a tabela 0, adicionando uma regra com prioridade *maxPrio* que redireciona os primeiros 128 bytes da mensagem para a porta correspondente ao controlador. Assim, o controlador pode identificar os dados do cabeçalho do pacote sem lidar com o payload, tomando decisões de roteamento ou configuração conforme necessário.

A rotina de reconfiguração e roteamento é executada com base no algoritmo $\mathcal{A}$ em uso. Para o caso estudado, o algoritmo empregado foi o [13]. Quando o controlador recebe uma notificação de que uma nova mensagem entrou na rede, ele realiza uma verificação para determinar se é necessária uma reconfiguração. Caso contrário, o switch é autorizado a encaminhar a mensagem para o próximo nó diretamente.

Se a reconfiguração for necessária, o controlador avalia o tipo de ajuste requerido e verifica se os clusters de switches necessários para estabelecer as novas conexões estão desbloqueados, garantindo que a estrutura em árvore não seja comprometida. Em caso positivo, é enviada uma mensagem de travamento para os switches envolvidos na reconfiguração. Esse travamento persiste pelo período necessário para que a mensagem alcance seu destino ou seja encaminhada para fora do conjunto de nós que participou da rotação.

Esse mecanismo foi adaptado do trabalho [14], de forma que, mesmo sem o uso de *time slots* discretos – dado que o simulador NS-3 trata o tempo como uma variável de ponto flutuante –, a consistência dos algoritmos é preservada. Assim, o processo de reconfiguração ocorre em cinco etapas principais: notificação do controlador, decisão sobre a reconfiguração, travamento dos clusters envolvidos, roteamento da mensagem e destravamento dos clusters.

D. Arquivo de entrada e geração de tráfego

A definição dos parâmetros de simulação são obtidas a partir de um arquivo de entrada contendo o número de nós (N), o número de mensagens ($|\sigma|$), e uma sequência de pares (s, t) que correspondem as mensagens. Ao finalizar a leitura desse arquivo é então gerado uma sequência de números de ponto flutuante correspondente ao momento da simulação em que cada mensagem será enviada do seu respectivo nó fonte (s). Essa distribuição de tempos segue uma Poisson com parâmetro T. A sequência de mensagens é mantida conforme descrita no arquivo de entrada, sendo que cada mensagem recebe um tempo de envio maior ou igual ao da mensagem anterior.

A partir dessa definição sequencial são agendados eventos utilizando o *Simulator* do NS-3 gerando um pacote com destino para o IP correspondente ao nó t. Cada nó recebe também um soquete preparado para escutar mensagens direcionadas para ele, registrando assim, o tempo de chegada da mensagem.

E. Geração da topologia

A construção dos clusters de switches leva em consideração a arquitetura da Fig. 1 onde há conjuntos de switches agrupados conforme quais clusters de ToRs eles interconectam.

Algorithm 1 CreateOpticNetTopology

Require: $p \geq 4$
Require: $N : log_p(N) \in \mathbb{N}$
  $C \leftarrow \lceil \frac{N}{p} \rceil$
  $nodeClusters \leftarrow listofToRnodes$
  buildClusters($nodeClusters, p$)
  buildBST($nodeClusters$)

Considerando clusters de nós $n, m \in N, n < m$ existem UnitClusters que são compostos por switches que realizam as conexões entre nós $x, y \in N_n$, ou seja intra-cluster, e os UnionClusters que são compostos por switches que realizam as conexões entre nós $x \in N_n, y \in N_m$, ou seja inter-cluster. Essa montagem é realizada pelo 2, em que os nós são atribuídos a clusters de switches e vinculados às portas de forma ordenada. Logo em seguida. durante a construção da representação abstrata da árvore da rede são gerados os matchings internos seguindo o modelo proposto pelo ToR-matching-ToR, gerando emparelhamentos entre as portas do switch.

Algorithm 2 buildCluster

Require: $p \geq 4$
  $unitClusters \leftarrow \emptyset$
  $unionClusters \leftarrow \emptyset$
  for $n \in nodeClusters$ **do**
    $unitClusters \leftarrow cluster_{nn}$
  end for
  for $n, m \in nodeClusters, n < m$ **do**
    $unionClusters \leftarrow cluster_{nm}$
  end for

F. Visão geral da utilização

Para utilizar o módulo de simulação de SDN é necessário adaptar os scripts de simulação conforme a necessidade do teste. Os passos para realizar essa adequação são

1) Criar o arquivo de sequência de mensagens conforme descrito na subseção III-D.
2) Adaptar template para gerar topologia desejada alterando o arquivo descrito na subseção III-E.

A configuração do ToR é realizada com base nos atributos definidos no script de simulação. Para instanciar o objeto é necessário atribuir o IP e o MAC do nó vinculado ao ToR além de conectar o ToR ao seu nó antes de conectá-lo aos outros switches, pois a porta exclusiva do terminal é dependente da ordem de conexão.

Para implementar variações do controlador de topologia da rede é necessário extender o controlador base do módulo alterando a forma como as regras de geração de resposta e encaminhamento de pacotes são construídas. Assim, os comandos *dpctl* devem ser adaptados conforme descrito na documentação [23] adicionando as alterações das tabelas de fluxo conforme necessário para gerar as rotas entre os nós comunicantes.

IV. Avaliação

O template de simulação utilizou uma implementação do SplayNet [13] com custo 1 para roteamento, levando em consideração cada reconfiguração com custo $\alpha = 1$. O algoritmo foi avaliado em três cenários com um número $N = 128$ de nós terminais, gerando um tráfego de 10000 mensagens com o tempo de emissão segundo uma distribuição de Poisson, para o *Bursty* e o *Projector*, e uma distribuição normal para o dataset *Normal*.

- **Bursty**, apresenta alta localidade temporal e baixa localidade não temporal, foi gerado utilizando o método descrito em [24].
- **Projector** [25], apresenta alta localidade não temporal e baixa localidade temporal, coletado a partir de uma coleta de $n = 128$ nós ToR selecionados aleatóriamente dentro de uma distribuição de probabilidade de comunicação de 8367 pares.
- **Normal**, apresenta um equilíbrio entre localidade temporal e não temporal.

Foram coletadas as métricas definidas na subseção II-G além dos tempos de envio b_i e chegada e_i. As entradas seguem as distribuições descritas em [24], [25] para que o tráfego seja semelhante ao de um datacenter com diferentes propósitos.

Os resultados, como pode ser visto na Fig. 6 demonstram que há um gargalo que resulta em um atraso das mensagens, o que é esperado em um ambiente real. Além disso, devido à forma como é configurada a distribuição de liberação das mensagens, limitando a janela a um curto espaço de tempo, como no caso do bursty, ou distribuindo mais ao longo do tempo, como no cenário da normal afeta o número de reconfigurações. Também a localidade espacial influencia o número de roteamentos que serão realizados, que também é afetado pelo

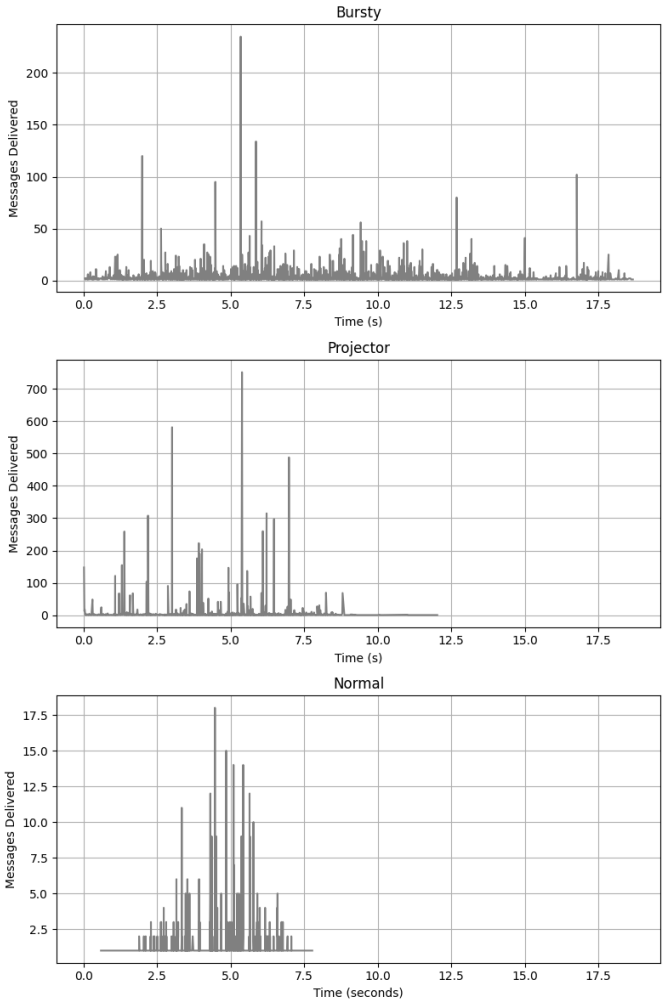

Fig. 4. Throughput das mensagens ao longo do tempo de simulação.

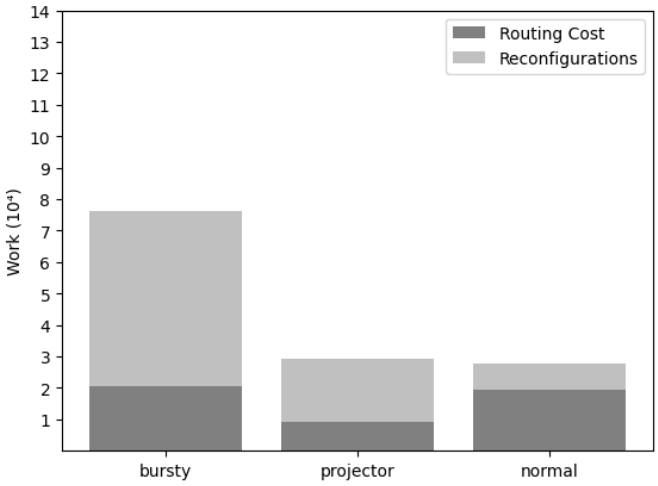

Fig. 5. Custo total de trabalho para $n = 128$ e 10.000 mensagens.

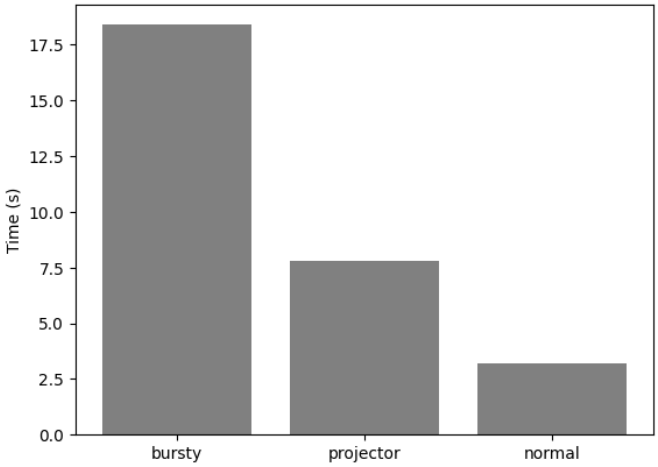

Fig. 6. Makespan calculado a partir das mensagens enviadas e recebidas.

tempo de intervalo entre as checagens pela liberação dos clusters envolvidos nas reconfigurações necessárias e quais reconfigurações estarão disponíveis, levando a resultados levemente diferentes dos vistos em [14].

Além disso, considerando as métricas definidas na subseção II-G, foi realizado um quadro comparativo em que é possível ver que apesar de o cenário Bursty possuir um custo de roteamento semelhante ao da distribuição Normal, o Makespan apresentado é o maior, levando a entender que possivelmente houveram gargalos com um número grande de reconfigurações, minimizando o número de roteamentos necessários havendo em contrapartida um tempo maior de entrega.

V. Tutorial de uso

Para instalar o módulo é necessário antes instalar os requerimentos mínimos que são o código do simulador NS-3 e o módulo com a implementação do OpenFlow 1.3. Os passos a seguir foram testados com *Ubuntu 22.04.04 LTS*.

A. Configurando o ambiente

Antes de iniciar o uso do módulo é necessário garantir que os requerimentos mínimos estão instalados no sistema.

```
$ sudo apt install g++ python3 cmake ninja-build git
$ sudo apt install make pkg-config libtool libbost-dev
```

B. Configurando o simulador

Clone o código do simulador NS-3, o módulo foi validado na versão ns-3.39.

```
$ git clone https://gitlab.com/nsnam/ns-3-dev.git
$ cd ns-3-dev
$ git checkout -b ns-3.39 ns-3.39
```

Então baixe o módulo *OFSwitch13* dentro do diretório contrib/. Foi utilizada a versão 5.2.2 durante a validação.

```
$ cd contrib/
$ git clone https://github.com/ljerezchaves/ofswitch13.git
$ cd ofswitch13 & git checkout 5.2.2
```

O módulo *OFSwitch13* requer um patch [21] aplicado nos componentes utilizados pelo switch OpenFlow.

```
$ cd ../../
$ patch -p1 < contrib/ofswitch13/utils/ofswitch13-3_39.patch
```

Após a conclusão dessas etapas o código do módulo pode ser clonado do repositório

```
$ git clone git@github.com:davilbs/light-flow.git
```

A partir desse ponto o simulador está pronto para ser configurado para compilação. O processo inclui o download e compilação da biblioteca *BOFUSS* necessária para comunicação entre o controlador e o switch. Para configurar:

```
$ ./ns3 configure --enable-examples
```

Verifique que ao configurar o simulador foi habilitada a feature *ns-3 OFSwitch13 integration*. Então compile o simulador:

```
$ ./ns3 build
```

C. Utilização do módulo

Dentro do diretório do módulo há um exemplo com o nome de *bst-traffic.cc* onde foi implementada a construção da topologia OpticNet [14] utilizando os controladores definidos na seção III. O código consiste em partes adaptáveis, e serve como base para construção de outras topologias a partir da lógica implementada para o OpticNet. Para executar o exemplo com um número p de portas em cada switch e lendo a sequência de mensagens a partir do arquivo *message-sequence.txt* basta usar o simulador com os parâmetros definidos como a seguir

```
$ ./ns3 run "bst-traffic --verbose --p=8
--filename=message-sequence.txt"
```

Ao final da execução são gerados dois arquivos de saída no formato .csv *mensagens.csv* contendo duas colunas que correspondem respectivamente a b_i e e_j, a linha não corresponde, necessariamente à mesma mensagem. E *resultado.csv* contendo o Makespan e o custo de roteamento.

Os componentes *TorController* e *FlatController* podem ser instanciados de forma individual, configurando o objeto separadamente. O NS-3 utiliza uma estrutura com *helpers* que seguem o padrão de projeto Factory e Decorator, onde os atributos do objeto são definidos de forma a produzir várias instâncias dele, porém ao instanciar um objeto podem ser adicionados novos atributos e funcionalidades acoplando outros objetos por meio do uso dos *helpers*.

Para instanciar um objeto representando um ToR segue-se o fluxo a partir do *helper* provido pelo módulo *OFSwitch13* chamado *OFSwitch13InternalHelper* da seguinte forma:

```
Ptr<OFSwitch13InternalHelper> torHelper =
                CreateObject<OFSwitch13InternalHelper>();
Ptr<Node> torNode = CreateObject<Node>();
\\ Conectar o torNode ao node host e aos switches
\\ inter/intra cluster
Ptr<Node> torControllerNode = CreateObject<Node>();
Ptr<TorController> torController =
                CreateObject<TorController>();
torController->SetAttribute("HostMacAddr",
                            Mac48Address::Allocate());
torController->SetAttribute("HostIpAddr",
                            Ipv4Helper.Assign());
torController->SetAttribute("HostPort", port);
torHelper->InstallController(torControllerNode,
                torController);
torHelper->InstallSwitch(torNode, torPorts);
torHelper->CreateOpenFlowChannels();
```

Nesse exemplo, o nó *torNode* hospeda um switch OpenFlow coordenado pelo controlador instalado no nó *torControllerNode*. Ao final o *helper* gera os canais OpenFlow e a tabela de fluxo para o switch, permitindo a divisão entre plano de controle e plano de dados, conforme especificado para SDNs. Para o switch intra/inter cluster o processo é semelhante, porém para configurar o controlador é necessário definir as conexões de forma a estabelecer a relação de clusters conforme descrito na Fig 1 permitindo a utilização das reconfigurações sem conflito na representação da estrutura ToR-matching-ToR. No exemplo *bst-traffic.cc* há uma implementação de um método auxiliar *makeClusters* que realiza a construção dos cluster a partir dos valores informados para parâmetros de número de portas p, além do número n de nós da rede.

```
Ptr<OFSwitch13InternalHelper> switchHelper =
                CreateObject<OFSwitch13InternalHelper>();
Ptr<Node> switchNode = CreateObject<Node>();
\\ Conectar o switchNode ao nodes correspondentes aos ToR
\\ da rede conforme a topologia OpticNet
Ptr<Node> switchControllerNode = CreateObject<Node>();
Ptr<FlatController> switchController =
                CreateObject<FlatController>();
switchController->SetClusters(nodes, nodeClusters,
                    unitClusters, unionClusters);
switchHelper->InstallController(switchControllerNode,
                switchController);
switchHelper->InstallSwitch(switchNode, switchPorts);
switchHelper->CreateOpenFlowChannels();
```

VI. Conclusão

A implementação final deste projeto seguiu o cronograma proposto, completando as etapas iniciais de desenvolvimento do módulo para simulação de switches ópticos em redes auto-configuráveis no simulador NS-3. Essa fase envolveu a configuração dos switches ToR e inter/intra-cluster, com a criação de controladores OpenFlow adaptativos para garantir que a rede fosse capaz de se ajustar dinamicamente às mudanças no tráfego.

Além disso, foi estruturada a leitura de arquivos de entrada, permitindo a definição dos parâmetros de simulação e a geração de tráfego a partir de uma sequência de mensagens com tempos de envio modelados segundo distribuições específicas. A montagem do template para a construção da topologia da rede também foi concluída, estabelecendo os links entre os nós e permitindo a simulação de diferentes topologias de rede, incluindo a representação de uma árvore geradora.

A coleta de dados foi realizada com base nas métricas de desempenho, como Makespan, custo de roteamento e o tempo de transmissão das mensagens. Esses resultados demonstraram que o módulo desenvolvido é capaz de simular redes com comportamentos dinâmicos e ajustáveis, proporcionando uma análise detalhada do desempenho de algoritmos de reconfiguração em diferentes cenários de tráfego.

Com isso, o projeto foi concluída com êxito, permitindo validar as funcionalidades básicas de redes auto configuráveis e fornecer uma base sólida para futuras melhorias. Como próximos passos, sugere-se a expansão do módulo para incorporar novos algoritmos de reconfiguração, bem como a avaliação de redes mais complexas e de maior escala, de modo

a explorar todo o potencial de adaptabilidade e eficiência das redes auto-configuráveis.

REFERENCES

[1] C. Avin and S. Schmid, "Toward demand-aware networking: a theory for self-adjusting networks," *SIGCOMM Comput. Commun. Rev.*, vol. 48, no. 5, p. 31–40, jan 2019. [Online]. Available: https://doi.org/10.1145/3310165.3310170

[2] M. Noormohammadpour and C. S. Raghavendra, "Datacenter traffic control: Understanding techniques and tradeoffs," *IEEE Communications Surveys Tutorials*, vol. 20, no. 2, pp. 1492–1525, 2018.

[3] C. Fuerst, S. Schmid, L. Suresh, and P. Costa, "Kraken: Online and elastic resource reservations for multi-tenant datacenters," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, 2016, pp. 1–9.

[4] H. Ballani, P. Costa, R. Behrendt, D. Cletheroe, I. Haller, K. Jozwik, F. Karinou, S. Lange, B. Thomsen, K. Shi, and H. Williams, "Sirius: A flat datacenter network with nanosecond optical switching," in *SIGCOMM*. ACM, August 2020. [Online]. Available: https://www.microsoft.com/en-us/research/publication/sirius-a-flat-datacenter-network-with-nanosecond-optical-switching/

[5] L. Poutievski, O. Mashayekhi, J. Ong, A. Singh, M. Tariq, R. Wang, J. Zhang, V. Beauregard, P. Conner, S. Gribble, R. Kapoor, S. Kratzer, N. Li, H. Liu, K. Nagaraj, J. Ornstein, S. Sawhney, R. Urata, L. Vicisano, K. Yasumura, S. Zhang, J. Zhou, and A. Vahdat, "Jupiter evolving: Transforming google's datacenter network via optical circuit switches and software-defined networking," in *Proceedings of ACM SIGCOMM 2022*, 2022.

[6] A. D. Ferguson, S. Gribble, C.-Y. Hong, C. Killian, W. Mohsin, H. Muehe, J. Ong, L. Poutievski, A. Singh, L. Vicisano, R. Alimi, S. S. Chen, M. Conley, S. Mandal, K. Nagaraj, K. N. Bollineni, A. Sabaa, S. Zhang, M. Zhu, and A. Vahdat, "Orion: Google's Software-Defined networking control plane," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, 2021, pp. 83–98.

[7] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, "Helios: a hybrid electrical/optical switch architecture for modular data centers," in *Proceedings of the ACM SIGCOMM 2010 Conference*, ser. SIGCOMM '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 339–350. [Online]. Available: https://doi.org/10.1145/1851182.1851223

[8] C. Avin and S. Schmid, *ReNets: Statically-Optimal Demand-Aware Networks*, 2021, pp. 25–39. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9781611976489.3

[9] C. Griner, J. Zerwas, A. Blenk, M. Ghobadi, S. Schmid, and C. Avin, "Cerberus: The power of choices in datacenter topology design - a throughput perspective," *SIGMETRICS Perform. Eval. Rev.*, vol. 50, no. 1, p. 99–100, jul 2022. [Online]. Available: https://doi.org/10.1145/3547353.3522635

[10] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "Bcube: a high performance, server-centric network architecture for modular data centers," in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, ser. SIGCOMM '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 63–74. [Online]. Available: https://doi.org/10.1145/1592568.1592577

[11] K.-T. Foerster, M. Pacut, S. Schmid, and E, "On the complexity of non-segregated routing in reconfigurable data center architectures," *SIGCOMM Comput. Commun. Rev.*, vol. 49, no. 2, p. 2–8, may 2019. [Online]. Available: https://doi.org/10.1145/3336937.3336939

[12] C. Avin, C. Griner, I. Salem, and S. Schmid, "An online matching model for self-adjusting tor-to-tor networks," *ArXiv*, vol. abs/2006.11148, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:219955787

[13] S. Schmid, C. Avin, C. Scheideler, M. Borokhovich, B. Haeupler, and Z. Lotker, "Splaynet: Towards locally self-adjusting networks," *IEEE/ACM Transactions on Networking*, vol. 24, no. 3, pp. 1421–1433, 2016.

[14] C. A. Caldeira, O. A. d. O. Souza, O. Goussevskaia, and S. Schmid, "Optical self-adjusting data center networks in the scalable matching model," *IEEE Transactions on Cloud Computing*, pp. 1–12, 2024.

[15] "Ns-3 simulator," https://www.nsnam.org/, [Online; accessed 25-July-2024].

[16] K. Benzekki, A. E. Fergougui, A. E. Elalaoui, and L. Toto, "Software-defined networking (sdn): a survey," *Security and Communication Networks*, vol. 9, no. 18, pp. 5803–5833, 2016. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/sec.1737

[17] H. Abdelgader Eissa, K. A. Bozed, and H. Younis, "Software defined networking," in *2019 19th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*, 2019, pp. 620–625.

[18] I. H. S, Y. R, and Y. V, "Implementing openflow, exploring the present and future software-defined networks ecosystem," in *2023 International Conference on Sustainable Communication Networks and Application (ICSCNA)*, 2023, pp. 280–285.

[19] W. M. Mellette, R. McGuinness, A. Roy, A. Forencich, G. Papen, A. C. Snoeren, and G. Porter, "Rotornet: A scalable, low-complexity, optical datacenter network," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 267–280. [Online]. Available: https://doi.org/10.1145/3098822.3098838

[20] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, p. 123–137, aug 2015. [Online]. Available: https://doi.org/10.1145/2829988.2787472

[21] "Openflow 1.3 module," https://www.lrc.ic.unicamp.br/ofswitch13/, [Online; accessed 25-July-2024].

[22] E. L. Fernandes, E. Rojas, J. Alvarez-Horcajo, Z. L. Kis, D. Sanvito, N. Bonelli, C. Cascone, and C. E. Rothenberg, "The road to bofuss: The basic openflow userspace software switch," *Journal of Network and Computer Applications*, p. 102685, 2020.

[23] "Dpctl docs," https://github.com/CPqD/ofsoftswitch13/wiki/Dpctl-Flow-Mod-Cases, [Online; accessed 25-July-2024].

[24] C. Avin, M. Ghobadi, C. Griner, and S. Schmid, "On the complexity of traffic traces and implications," in *Abstracts of the 2020 SIGMETRICS/Performance Joint International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 47–48. [Online]. Available: https://doi.org/10.1145/3393691.3394205

[25] M. Ghobadi, R. Mahajan, A. Phanishayee, N. Devanur, J. Kulkarni, G. Ranade, P. Blanche, H. Rastegarfar, M. Glick, and D. Kilper, "Projector: Agile reconfigurable data center interconnect," in *SIGCOMM 2016 - Proceedings of the 2016 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM 2016 - Proceedings of the 2016 ACM Conference on Special Interest Group on Data Communication. Association for Computing Machinery, Inc, Aug. 2016, pp. 216–229, publisher Copyright: © 2016 ACM.; 2016 ACM Conference on Special Interest Group on Data Communication, SIGCOMM 2016 ; Conference date: 22-08-2016 Through 26-08-2016.