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 crucial no suporte à crescente demanda por infraestrutura de nuvem e aplicações centradas em dados. No entanto, as topologias de rede tradicionais dentro dos data centers frequentemente sofrem com configurações estáticas que não se adaptam às demandas dinâmicas da rede, levando a ineficiências. Para enfrentar esses desafios, este artigo apresenta um novo módulo para o simulador NS-3, projetado para avaliar e melhorar o desempenho de topologias de rede reconfiguráveis em data centers. Ao implementar topologias auto configuráveis, buscamos aumentar a eficiência dos recursos e nos adaptar a padrões de tráfego variados. Nosso módulo facilita o desenvolvimento e teste de algoritmos de reconfiguração, simulando cenários de tráfego sintético e do mundo real. Métricas-chave, como makespan e custo de roteamento, são coletadas para fornecer insights sobre o desempenho dos algoritmos. Este trabalho contribui para o avanço da adaptabilidade e eficiência das redes de data centers, apoiando o desenvolvimento de redes autoadaptáveis.**

I. Introdução

Nos últimos anos, a importância dos datacenters tem crescido significativamente, impulsionada pelo aumento dos serviços que dependem de infraestrutura de nuvem e por aplicações cada vez mais centradas em dados [1], [2]. Os datacenters desempenham um papel crucial no suporte a uma ampla gama de aplicações, desde sistemas de e-commerce até serviços de streaming em tempo real.

Além disso, o avanço em aplicações distribuídas de aprendizado de máquina tem se mostrado um motor vital para o crescimento dos datacenters. Trabalhos recentes [3], [4] demonstram como essas aplicações se beneficiam da infraestrutura robusta dos datacenters para processamento e análise de grandes volumes de dados. No entanto, com a demanda crescente, aumenta também a pressão sobre os datacenters para utilizarem seus recursos de forma mais eficiente [5], [6].

Para atender a essa necessidade de eficiência, novas topologias de rede foram propostas, buscando otimizar o uso dos recursos e melhorar o desempenho das comunicações dentro dos datacenters [7]–[9]. No entanto, as topologias tradicionais apresentam limitações, como a natureza estática, que impede a adaptação dinâmica às mudanças nas necessidades da rede [10], [11].

Como solução, propõe-se o desenvolvimento de topologias auto configuráveis, capazes de se adaptar ao tráfego da rede em tempo real [12], [13]. Essas topologias prometem aumentar a eficiência do uso dos recursos e melhorar o desempenho geral da rede. Contudo, há uma carência de recursos adequados para o desenvolvimento e teste de algoritmos de reconfiguração [1].

Neste contexto, o simulador NS-3 surge como uma solução viável, oferecendo um ambiente flexível e escalável para o desenvolvimento e a análise de algoritmos de reconfiguração [14]. Para validar o módulo proposto, foram simulados cenários de tráfego sintético e real, com a coleta de métricas ao longo da simulação para avaliar o desempenho dos algoritmos. Este artigo apresenta uma abordagem inovadora para a validação de algoritmos de reconfiguração de topologias em datacenters, contribuindo para o avanço da eficiência e da adaptabilidade das infraestruturas de nuvem.

II. Referencial

A. Topologia da rede

A rede é constituída por duas categorias de comutadores Top-of-Rack (ToR) switches e Switches inter-cluster. 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.

Foi implementada a topologia de rede BCube conforme descrito por Chuanxiong Guo et al. em [10]. A rede consiste em estruturas recursivas, se iniciando em um cluster $BCube_0$, onde há um switch com n portas conectado a n ToR switches. Para os $BCube_k$ onde $k > 0$ os ToRs são conectados entre clusters diferentes utilizando um switch inter-cluster representando o $BCube_{k+1}$. Essa alta conectividade permite que o BCube exemplifique a funcionalidade do modelo proposto em um cenário semelhante ao de um datacenter real.

B. Estrutura SDN

Uma rede definida por software (SDN) é um paradigma de construção de redes de computadores em que o processo de 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). [15], [16] 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 [17] na versão 1.3. Esse protocolo define a

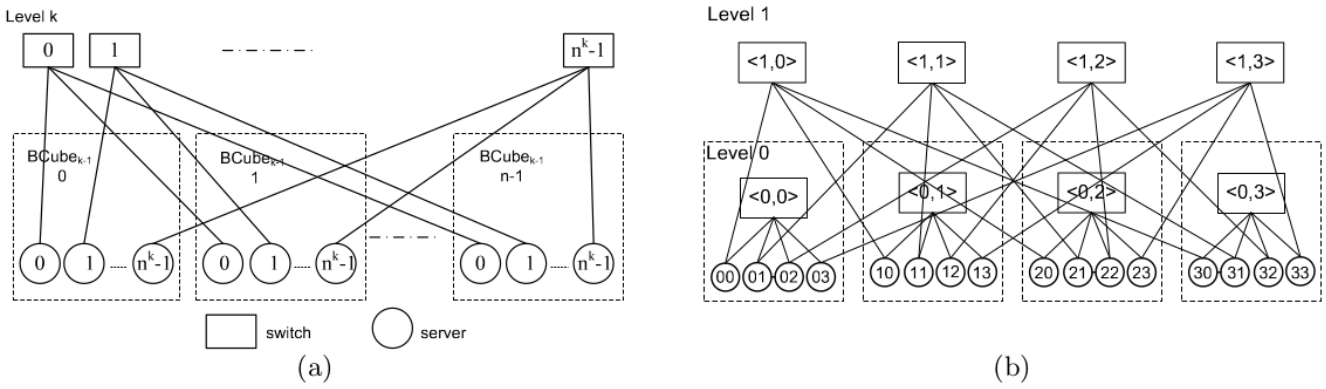

Fig. 1. (a) BCube é um estrutura em níveis. Um $BCube_k$ é construído a partir de $nBCube_{k-1}$ e n^k switches com n portas. (b) Exemplo de $BCube_1$ com $n = 4$.

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 [18]. 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 [19].

D. Simulador ns-3

Considerando os requisitos para construção do ambiente virtual de simulação foi escolhido o NS-3 [14]. 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.

E. Módulo OfSwitch13

Para complementar o NS-3 com funcionalidades requeridas para simulação de SAN o módulo OFSwitch13 [20] 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 [21], 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. Porém, o módulo apresenta uma limitação que é a mesma apresentada

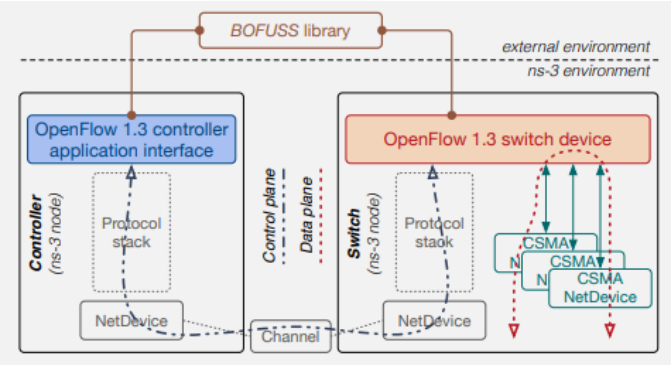

Fig. 2. Visão geral do módulo OFSwitch13 utilizado

pelo simulador base que é a ausência de um mecanismo para resolução de ciclos na topologia da rede, dessa forma apenas extender o comportamento da classe disponibilizada não é o suficiente para simular redes de datacenter como proposto, requerendo uma adaptação do processo de montagem topológica como será descrito na subseção III-E.

F. Métricas (Makespan, 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 duas métricas como definidas em [22], o Makespan e o custo de roteamento. Para o Makespan, considere uma sequência σ de m mensagens com origem s e destino d, uma topologia em árvore T_0. Para uma mensagem $\sigma_i(s,d) \in \sigma$, 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

$$Makespan(T_0, \sigma) = \max_{1 \leq i \leq m} e_i - \min_{1 \leq i \leq m} b_i \qquad (1)$$

Considerando essa sequência de mensagens, temos que o custo de roteamento será de $d_{e_i}(s,d)$, o comprimento do caminho $P_{e_i}(s,d)$ na árvore resultante T_{e_i}, ou seja, após a entrega de σ_i, correspondendo ao número de encaminhamentos do pacote na rede. O custo de roteamento é dado por

$$D(T_0, \sigma) = \sum_{i=1}^{m} (d_{e_i}(s,d)(\sigma_i) + 1) \qquad (2)$$

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. As tarefas realizadas nessa etapa estão descritas na tabela I marcadas como POC I.

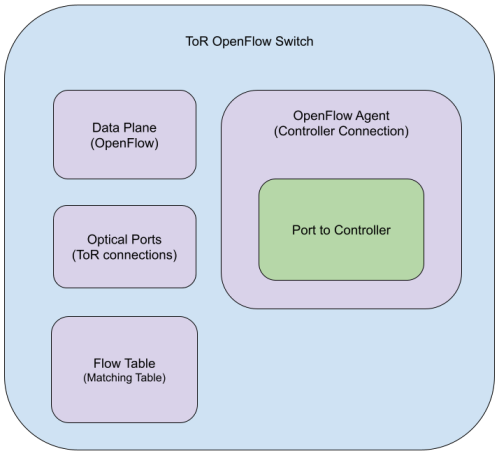

Fig. 3. Arquitetura da implementação do switch utilizando um controlador OpenFlow para construir a tabela de resolução de endereços. O comportamento é definido pelo controlador, sendo necessárias adaptações para o ToR e o switch intra/inter cluster.

B. Classe do 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 [21] 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 pa-

cotes 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 [20]. 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 entrada com prioridade com valor *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 ter que lidar com o payload, e tomar a decisão de acordo.

Ao inicializar o controlador, é passada a matriz de adjacência gerada a partir da construção da árvore geradora do grafo da topologia, conforme descrito na subseção III-E. A partir dessa matriz o controlador popula as tabelas L2 dos switches inter e intra-cluster mapeando os endereços IP aos MAC correspondentes dos ToRs e seus respectivos servidores. A árvore é navegada a partir de cada switch para que as mensagens codificadas utilizando o *dpctl* mapeiem o fluxo para as portas corretas.

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 sem loops

Para inicializar as estruturas é utilizado o laço no algoritmo 1 de forma a popular a matriz de adjacência conforme os parâmetros N (número de nós na rede). Essa etapa pode ser substituída para gerar outros tipo de de topologia, sendo necessário apenas adequar os parâmetro de execução do template para a topologia escolhida.

Com base na matriz de adjacência, a Spanning Tree é construída de forma a remover os loops da rede prevenindo broadcast storms de ocorrerem. Para gerar a árvore geradora do grafo topológico é escolhida uma raiz qualquer a partir da qual é percorrido o grafo original com um DFS, criando os links entre os nós vizinhos visitados. Isso decorre da ausência do STP (Spanning Tree Protocol) ou do OSPF (Open Shortest Path First) no ambiente do NS-3, que gera problemas em uma topologia densamente conectada como BCube.

Algorithm 1 CreateBCubeTopology

Require: $k \geq 1$
Require: $N : log_k(N) \in \mathbb{N}$
  $maxLevel \leftarrow log(N)/log(k) - 1$
  $M \leftarrow adjacencymatrix$
  $ST \leftarrow spanningtreematrix$
  while $i < (N/k)$ **do**
    buildBCubeLevel($maxLevel, k, i, M$)
  end while
  buildSpanningTree($root, M, ST$)

A construção recursiva leva em consideração a posição relativa dos switches inter e intra-cluster na matriz de adjacência para mapear a porta ao nó correspondente.

Algorithm 2 buildBCubeLevel

Require: $k \geq 1$
  $M \leftarrow adjacencymatrix$
  $step \leftarrow Level^k$
  $first \leftarrow (Index/step) * (step * k) + (Index\%step)$
  $last \leftarrow ((Index/step) + 1) * (step * k) + (Index\%step)$
  if $currentLevel > 0$ **then**
    buildBCubeLevel($Level - 1, k, Index, M$)
  end if
  for $j = first; j < last; j + step$ **do**
    $M(currentIdx, j) \leftarrow 1$
  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. Experimento

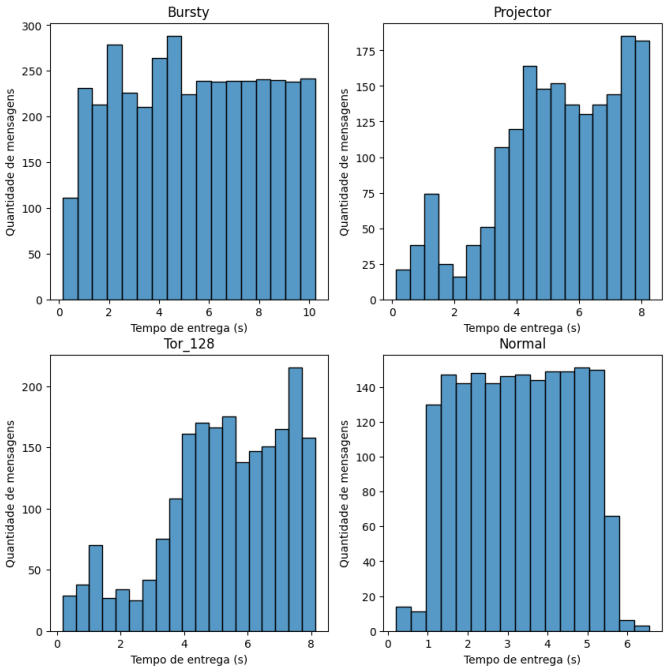

Fig. 4. Tempos de chegada das mensagens nos nós finais. Algumas mensagens foram perdidas durante o trânsito resultando em um número menor de mensagens alcançando o destino.

O template de simulação foi avaliado em quatro cenários com um número $N = 128$ de nós terminais, gerando um tráfego de 10000 mensagens. Foram coletadas as métricas definidas na subseção II-F além de os 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 figura 4 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 a árvore geradora é construída pode haver mais, como visto para o Projector ou menos perda de mensagens, como visto para o cenário Bursty.

Além disso, considerando as métricas definidas na subseção II-F, foi realizado um quadro comparativo em que é possível ver que apesar de o cenário Bursty possuir menor custo de roteamento o Makespan apresentado é o maior, levando a entender que possivelmente houveram gargalos com atrasos mais constantes, permitindo uma menor perda de pacotes com um tempo maior de entrega.

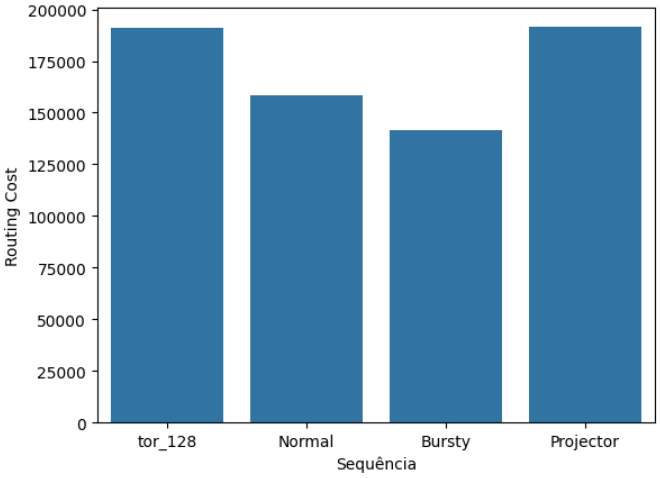

Fig. 5. Custo total de roteamento calculado a partir da matriz de adjacência gerada para a topologia.

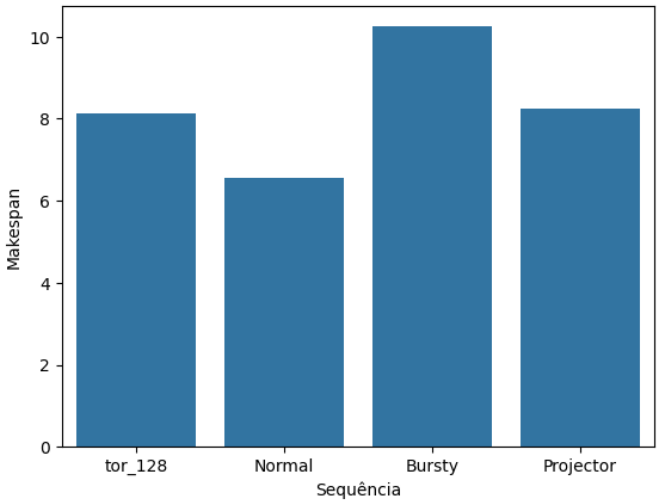

Fig. 6. Makespan calculado a partir das mensagens enviadas e recebidas.

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 [20] 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 *bcube-traffic.cc* onde foi implementada a construção da topologia bcube 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 BCube. Para executar o exemplo com um número k 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 "bcube-traffic --verbose --k=12
--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 *BCubeController* 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 a matriz de adjacência referente à árvore geradora da rede. No exemplo *bcube-traffic.cc* há uma implementação de um método auxiliar *makeSpanningTree* que realiza a construção da matriz a partir da matriz de adjacência da topologia escolhida.

```
Ptr<OFSwitch13InternalHelper> switchHelper =
                CreateObject<OFSwitch13InternalHelper>();
Ptr<Node> switchNode = CreateObject<Node>();
\\ Conectar o switchNode ao nodes correspondentes aos ToR
\\ da rede segundo a matriz da spanning tree
Ptr<Node> switchControllerNode = CreateObject<Node>();
Ptr<BCubeController> switchController =
                CreateObject<BCubeController>();
switchController->SetSpanningTree(spanningTree, numNodes);
switchController->SetMapping(switchConnections);
switchHelper->InstallController(switchControllerNode,
                switchController);
switchHelper->InstallSwitch(switchNode, switchPorts);
switchHelper->CreateOpenFlowChannels();
```

VI. Conclusão

A implementação inicial concluída seguiu a primeira etapa do cronograma proposto I. Essa etapa incluiu a configuração dos switches ToR, inter e intra-cluster por meio do desenvolvimento de conroladores OpenFlow capazes de adaptar o comportamento ao esperado em uma rede com potenciais loops. A estruturação da leitura de um arquivo de entrada contendo os parâmetros para simulação de uma sequência bem definida de mensagems e, a partir dessa leitura realizar a geração de tráfego conforme uma distribuição de tempos de envio. Também foi concluída montagem de um template para construção da topologia, estabelecendo os links entre os nós e gerando uma matriz de adjacência correspondente à uma árvore geradora da rede. Por fim, a coleta de resultados parciais por meio do Makespan, custo de roteamento e de dados de tráfego a partir da coleta de tempos de saída e chegada das mensagens.

A. Próximos passos (POC II)

Para o trabalho futuro em POC II como descrito na tabela I, ainda é necessário implementar a funcionalidade de reconfigurações nos switches e desenvolver um controlador com um algoritmo de reconfiguração. Além disso, é preciso implementar e testar os controladores de topologia, bem como ajustar os switches para adequar seu comportamento às reconfigurações de circuito. Novas métricas, como o custo

TABLE I
CRONOGRAMA DE ATIVIDADES DO PROJETO

Etapa	Tarefa
POC I	Leitura do arquivo de entrada
POC I	Implementação do template de construção de topologia
POC I	Implementação da topologia BCube
POC I	Implementação do controlador do ToR switch
POC I	Implementação do controlador dos switches inter/intra cluster
POC I	Coleta de dados de tráfego (tempo de envio e chegada) durante a simulação
POC I	Cálculo das métricas Makespan e Custo de Roteamento
POC I	Geração do tráfego conforme distribuição de probabilidade mantendo a ordem sequencial
POC I	Construção das tabelas L2 dos switches
POC II	Adaptação do ToR e inter/intra switches para considerar funcionalidades de switch de circuito
POC II	Implementar topologia de rede flat separada em clusters
POC II	Adicionar tabelas de matching para estrutura tor-matching-tor
POC II	Implementar controlador com algoritmo de reconfiguração

amortizado, serão adicionadas, sendo especialmente relevantes para o cenário de interesse, que são as Redes Auto Configuráveis (SAN).

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

[15] 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

[16] 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.

[17] 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.

[18] 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

[19] 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

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

[21] 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.

[22] O. Souza, C. Caldeira, and O. Goussevskaia, "Redes auto-reconfiguráveis randomizadas para comunicação distribuída em datacenters," in *Anais do XL Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. Porto Alegre, RS, Brasil: SBC, 2022, pp. 377–390. [Online]. Available: https://sol.sbc.org.br/index.php/sbrc/article/view/21184

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

[24] 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.

[25] 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