Um Modelo Formal de Propósito Geral para Simulações de Redes Definidas por Software

Rafael Souza[1], **Marcelo Santos**[1], **Braulio Mello**[2], **Stênio Fernandes**[1]

[1]Centro de Informática – Universidade Federal de Pernambuco (UFPE)
Caixa Postal 181 - CEP 89802-112 – Recife – PE – Brasil

[2]Ciência da Computação – Universidade Federal da Fronteira Sul (UFFS)
Chapecó, Brasil. Caixa Postal 181 - CEP 89802-112 Chapecó - SC - Brasil

{rrs4, mabs, sflf}@cin.ufpe.br, braulio@uffs.edu.br

Abstract. *Software Defined Networking (SDN) arises as a new paradigm for network management due to the clear separation between the control plane and the data plane. In this context, in order to measure which characteristics affect network performance is an arduous task considering different scenarios, several requirements, and different optimization objectives. Hence, we developed a model based on DEVS formalism that enables to execute SDN simulations quickly and in a flexible way. The proposed model makes it possible to analyze performance issues that may impact on SDN deployment. Moreover, it is also possible to assess optimization strategies that can be integrated into DEVS opening up the opportunity to validate other published papers.*

Resumo. *Redes Definidas por Software (SDN) permitem uma maior flexibilidade na gerência de fluxos de rede devido a clara separação entre o plano de controle e o plano de dados. Nesse contexto, mensurar as características que afetam, por exemplo, o desempenho da rede é uma tarefa árdua diante de diferentes cenários, requisitos e objetivos distintos de otimização. Assim, desenvolvemos um modelo baseado no formalismo de especificação de sistemas a eventos discretos (DEVS) para que seja possível simular cenários de rede SDN de forma flexível e rápida a fim de auxiliar no entendimento do funcionamento da rede e de aspectos que podem impactar no seu desempenho. Além disso, é possível ainda avaliar estratégias de otimização que podem ser integradas ao DEVS abrindo a oportunidade para validar diversos outros trabalhos.*

1. Introdução

Atualmente, o funcionamento de redes de computadores depende de milhões de dispositivos de comunicação, o que acarreta uma complexidade da rede. Consequentemente, há diversos desafios devido à dificuldade de realizar modificações no núcleo da rede ocasionando, por exemplo, o problema conhecido como ossificação da *Internet*. Diante desses problemas, bem como da necessidade de redução das despesas de capital (*Capital EXpenditure* -CapEX) e de custos operacionais (*Operational EXpenditure* - OpEX), diversas abordagens surgiram como uma possível solução para os desafios atuais enfrentados na *Internet*, entre as quais podemos destacar duas grandes vertentes: (1) evolucionária - *evolutionary* e (2) limpa - *clean slate* [Rexford and Dovrolis 2010]. Neste sentido, surgiram

as redes definidas por software (*Software Defined Networking* - SDN) que permitem desacoplar o plano de controle (lógico) do plano de dados, propiciando um cenário evolutivo das redes atuais.

Técnicas de Modelagem e Simulação (M&S) têm-se mostrado capazes de contribuir com a redução do esforço e custo para planos de avaliação de cenários de rede [Wainer 2009]. Todo o campo de redes de computadores passa pela análise/avaliação de desempenho para conceber novos cenários, arquiteturas, protocolos, entre outros. Assim, uma das contribuições deste trabalho é disponibilizar modelos hierárquicos e genéricos para que seja possível explorar eficientemente detalhes sobre o comportamento e estrutura de uma SDN. Para tal, é interessante utilizar mecanismo que permita acoplamento hierárquico para coexistência de modelos em representações mais simples de subsistemas. Os modelos devem considerar todas as interconexões e interações entre os mesmos, a fim de permitir um conjunto de modelos acoplados que representam o sistema complexo [Fishwick 1995]. Alguns trabalhos têm proposto técnicas de simulação para superar limites de escalabilidade [Curtis et al. 2011] [Yu et al. 2011] [Bari et al. 2013]. Em [Handigol et al. 2012] são apresentadas melhorias para o desenvolvimento e teste de novas aplicações do controlador por meio do Mininet, mas ainda tem limitações de escalabilidade, pois não pode lidar com grande quantidade de tráfego, isto porque a interface de auto-retorno (*loopback*) não é capaz de processá-lo em tempo hábil. Em [Gupta et al. 2013] foi proposto o fsSDN, uma ferramenta para criação de protótipos e avaliação de novas aplicações baseada em SDN. No entanto, nosso trabalho vai além de uma ferramenta especifica, por permitir que o modelo possa ser genérico o suficiente para abordar diversas características de SDN, baseado na robustez e grande escalabilidade do formalismo DEVS, o qual detalhamos na seção 3.

A construção de modelos em DEVS segue uma estrutura hierárquica e as transições de eventos e o avanço de tempo são formalmente definidos. Essas características fornecem vantagens para a modelagem e simulação, tais como: (i) facilidade para experimentação, (ii) testes, (iii) manutenção, e (iv) escalabilidade. Assim, o modelo proposto permite alterações de acordo com especificação, comportamento e métricas da rede em análise. As nossas contribuições são em duas vertentes: (i) construção de um modelo genérico para avaliação de desempenho de SDN, (ii) definição de uma metodologia para planejamento e avaliação de desempenho de SDN.

Dentro da IETF/IRTF, o *Software Defined Networking Research Group* (SDNRG) investiga SDN sob várias perspectivas com o objetivo de identificar as abordagens que podem ser definidas, implementadas e utilizadas no curto prazo, além de também identificar os futuros desafios de pesquisa. Em particular, as principais áreas de interesse incluem soluções escaláveis, abstrações, linguagens de programação e paradigmas particularmente úteis no contexto da SDN, bem como identificação de futuros desafios de pesquisa na área. Assim, acreditamos que criar uma modelo genérico que propicie a execução de simulações para investigar o comportamento de cenários complexos em SDN pode ser uma contribuição para o SDNRG que tem investigado, por exemplo, a performance de controladores SDN [Bhuvan et al. 2016a] [R. Gu et al. 2016] [Bhuvan et al. 2016b]. Além da comunidade da IETF/IRTF, consideramos uma contribuição para comunidade acadêmica que poderá utilizar um modelo existente para executar ou replicar algum experimento relacionado a SDN.

2. Fundamentação Teórica

2.1. Redes Definidas Por Software

Uma análise da arquitetura atual dos roteadores permite observar que se trata de um modelo formado basicamente por duas camadas bem distintas: o software de controle (*control plane*) e o hardware dedicado ao encaminhamento de pacotes (*data plane*) [Feamster et al. 2013]. SDN tem como base a dissociação dos planos de controle e de dados dos roteadores e *switches*, visando tornar os *switches* mais baratos, rápidos e flexíveis. SDN pode ser visto como um paradigma recente que promete fornecer um amplo controle sobre fluxos da rede, simplificando o gerenciamento através da utilização de um controlador externo. Desta forma, o tráfego pode ser explicitamente roteado e inspecionado por elementos de rede de controle com segurança e políticas de gerenciamento eficientes.

Desafios neste campo de pesquisa vêm do fato de que a lógica de projeto da arquitetura de uma SDN foi baseada em uma abordagem de cima para baixo (ou seja, da camada de plano de controle para os dispositivos da camada de plano de dados subjacente em uma infraestrutura), a fim de resolver problemas de eficácia ou eficiência da rede. No entanto, novos cenários de implantação de serviços SDN estão se tornando mais complexos e estão migrando para as infraestruturas distribuídas e heterogêneas, exigindo uma nova concepção lógica. Desafios de pesquisa para uma arquitetura SDN escalável podem ser abordados em vários aspectos, desde otimizações do plano de dados até a distribuição de funcionalidades do plano de controle. Como um novo paradigma para a computação em nuvem e ambientes de datacenters, mecanismos escaláveis que envolvem SDN são esperados para alavancar oportunidades de negócios potencialmente transformadoras.

2.2. Especificação Formal de Eventos Discretos

O formalismo de especificação de evento discreto (*Discrete Event System Specification* - DEVS) foi originalmente proposto por Zeigler [Zeigler et al. 2000] no final dos anos 70 para modelagem de sistemas de eventos discretos. Devido à sua capacidade de modelar sistemas complexos de maneira concisa e sem ambiguidades, o DEVS tem sido usado em diversas aplicações da engenharia (por exemplo: sistemas de manufatura, sistemas de comunicação, projeto de hardware) e ciência (por exemplo: biologia). Desde a formulação inicial de Zeigler, diversas variantes foram propostas para facilitar a modelagem e expandir as classes de sistemas que podem ser representadas pelo DEVS, tais como: *Classic DEVS System Specification*, *Parallel DEVS System Specification*, *Hierarchical Models*, *Object-Oriented Implementations of DEVS* [Zeigler et al. 2000]. Além de propostas de extensões para DEVS com o objetivo de simular sistemas com restrições temporais, tais como em [Mello and Wainer 2016].

Os modelos básicos DEVS, podem ser visualizados como caixas pretas que apenas interagem por meio de uma interface bem definida e hierárquico. O seu comportamento interno é definido por um conjunto de funções. Modelos ditos acoplados combinam os modelos básicos em uma estrutura hierárquica que define como ocorre a comunicação entre os modelos básicos. O comportamento de um sistema é descrito através das portas de entrada e saída dos modelos de seu estado e transições. O modelo mais básico, ou modelo atômico, é definido da seguinte forma [Zeigler et al. 2000]:

$$\text{MA} = (\text{X}, \text{Y}, \text{S}, \delta_{ext}, \delta_{int}, \lambda, t_a)$$

onde,

- X: é o conjunto de valores de entrada;
- Y: é o conjunto de valores de saída;
- S: é o conjunto de estados;
- δ_{ext}:Q x S $\rightarrow$ é a função de transição externa, onde $Q = (s, e)|s \in S, 0 = e = t_a(s))$ é o conjunto total de estados *e* é o tempo decorrido desde a última transição;
- δ_{int}:$S \rightarrow S$ é a função de transição interna;
- λ: $S \rightarrow Y$ é a função de saída;
- t_a: $S \rightarrow \Re_0^+ \bigcup \infty$ é a função de avanço de tempo (com $\Re_0^+ \bigcup \infty$) o conjunto de reais positivos com 0 e ∞.

Um conjunto de modelos atômicos conectados, forma o modelo acoplado. Assim, o DEVS permite que os modelos acoplados possam ser usados como modelo atômico dentro de um modelo maior, construindo dessa forma, uma estrutura hierárquica. O modelo acoplado de uma forma modular é definido formalmente da seguinte maneira [Zeigler et al. 2000]:

$$\text{MC} = (X, Y, D, \{M_i\}, \{I_i\}, \{Z_{i,d}\}, \text{Select})$$

onde,

- X: é o conjunto de valores de entrada;
- Y: é o conjunto de valores de saída;
- D: é o componente de referência do conjunto, de modo que para cada $d \in D$, é um modelo DEVS;
- M_i: índice do componente do modelo acoplado;
- I_i: é o conjunto de influência do modelo *i* e $\forall\ j \in I_i$;
- $Z_{i,j}$: $Y_i \rightarrow X_j$ tem *i* para *j* como função de transição;
- $Select$: é o seletor *tie-break*.

A Figura 1 ilustra a estrutura de um modelo DEVS, em que podemos interpretar os elementos da seguinte forma: a qualquer momento o sistema está em algum estado "S". Nesse caso, se não acontecer nenhum evento externo, o sistema ficará no estado "S"por um período de tempo t_a(s). Vale a pena ressaltar que t_a(s) é um número real, mas também pode ser um número entre 0 e ∞. Assim, na primeira situação, o sistema ficará no estado "S"para sempre, isso porque ele aguarda até que aconteça algum evento externo para que ocorra uma transição interna. Quando o tempo de *resting* finalizar, o tempo decorrido de *eleapsed time e* = t_a(s), o sistema vai apresenta as saídas nas portas *output*, por meio da função de λ(s) e, com isso, altera o estado através da função interna δ_{int}. Agora, se um evento (x $\in X^b$) externo acontecer antes de finalizar o tempo, o sistema fica em um estado (s,e) com $e \leq t_a$(s). O sistema vai executar a transição de estado, através da função externa δ_{ext} (s,e,x). Portanto, isso implica que a função de transição externa estabelece um novo estado para o sistema [Zeigler et al. 2000].

Em resumo, para construção de modelos em DEVS, primeiro se definem os modelos básicos a partir dos quais modelos maiores podem ser construídos. A partir disso, os modelos são conectados de modo a formar uma hierarquia. Teoremas, provas e detalhes adicionais sobre os formalismos podem ser obtidos em [Zeigler et al. 2000].

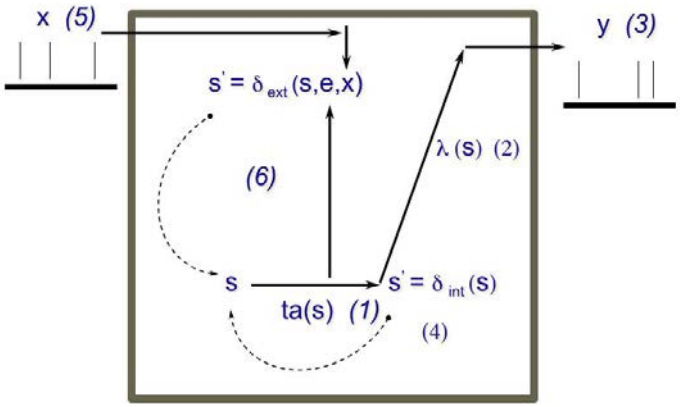


Figura 1. Estrutura de um modelo DEVS. [Zeigler et al. 2000]

3. DEVS na Simulação de Redes SDN

Este trabalho propõe um modelo genérico para representação do comportamento de redes SDN, baseado no formalismo DEVS. Existem outros formalismos na literatura que modelam e simulam sistemas de eventos discretos, como redes de Petri [Molloy 1982] [Murata 1989] e cadeias de Markov [Puterman 2014]. No entanto, esses formalismos têm o problema de explosão de espaço de estado que dificulta a modelagem dos sistemas grandes e complexos [Valmari 1998]. Além desse fator, a escolha do formalismo DEVS foi porque o mesmo traz simplicidade no desenvolvimento e diversas vantagens em representar o comportamento de redes SDN, bem como facilidade para especificação dos parâmetros do modelo de simulação de eventos discretos. Entre as vantagens, podemos citar as seguintes: fornece um *framework* formal para M&S; suporta uma completa capacidade de representação de sistema dinâmico; suporta hierárquica para o desenvolvimento modular do modelo; separa modelagem de simulação; deriva-se do formalismo genérico de sistemas dinâmicos; fornece acoplamento bem definido de componentes; suporta a construção hierárquica e reúso de modelos a partir de repositório.

Um outro ponto importante que ajudou na adoção do formalismo foi a disponibilidade de ferramentas em código aberto, como o CD++ Builder, que é um conjunto de ferramentas DEVS para modelagem e simulação que fornece uma biblioteca em C++ para especificar os modelos em vários formalismos DEVS (por exemplo. Cell DEVS, PDEVS) [Wainer 2009] [Wainer 2002]. Este trabalho adota a variante denominada Classic DEVS System Specification. As simulações podem ser realizadas localmente ou remotamente, através do envio de especificações do modelo para o servidor de simulação. O Kernel do CD++ Builder fornece um *plug-in* que permite edição de modelos, tanto textual quanto graficamente, bem como a visualização de resultados de simulação.

4. Modelos DEVS e Avaliação

4.1. Metodologia

Esta seção apresenta a metodologia concebida para avaliar o modelo em um cenário SDN simples. A execução é compreendida em 5 passos, que são: especificação do sistema, modelo atômico, modelo acoplado, construção final do modelo, e avaliação. A modelagem é feita por meio do formalismo DEVS, com uma estrategia hierárquica para representar o

modelo de desempenho genérico para arquiteturas SDN. Na figura 2, é descrito o processo realizado pela adoção do formalismo DEVS para modelagem hierárquica.

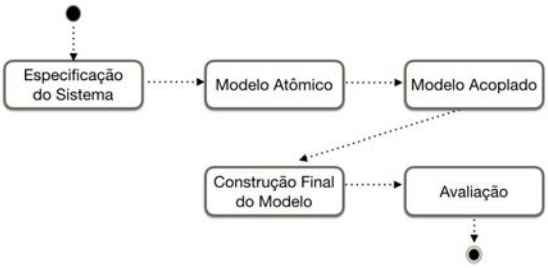


Figura 2. Metodologia para Avaliação de Redes SDN, usando o Modelo DEVS.

- Tem finalidade de compreender todos os parâmetros que são utilizados na avaliação do sistema, tais como a funcionalidade e interação entre o sistema e subsistema. Nesta fase, é importante identificar o problema e detalhes a serem analisados em um ambiente SDN;
- Modelo Atômico: Define os parâmetros do submodelo a serem analisados, a partir das informações do passo anterior são criados os modelos atômicos (submodelos) de SDN;
- Modelo Acoplado: É construída a lógica de relação e comunicação entre componentes (*switches* e controladores);
- Construção do Modelo Final: A partir deste modelo, é possível extrair as métricas desejadas. Este trabalho, adota métricas de *Flow Setup Time* (FST) e *Flow Count*;
- Avaliação: O modelo do sistema é avaliado pelas métricas e produz os resultados de interesse.

4.2. Descrição do Sistema

O cenário modelado compreende uma rede WAN, com *switches OpenFlow* [McKeown et al. 2008]. Além disso, há apenas um único controlador e consequentemente um único domínio, dado que o objetivo é apenas validar a usabilidade do modelo proposto. O controlador tem como responsabilidade a criação de caminhos entre os *switches* dentro do seu próprio domínio, onde periodicamente coleta o fluxo e estatísticas de portas a partir dos *switches*. Quando um novo fluxo chega em um *switch*, primeiro é verificado se já existe uma regra de encaminhamento para esse fluxo. Se a regra existe, então o pacotes desse fluxo são encaminhados sem haver nenhuma consulta ao controlador, seguindo, assim, a definição da regra existente. Caso contrário, o *switch* solicita ao controlador por meio de mensagem $Packet_In$ *Openflow* que seja informado como realizar o encaminhamento do fluxo, ocasionando um atraso no encaminhamento do fluxo até que se receba a resposta do controlador. O tempo necessário para realizar esta operação é conhecido como *flow setup time*.

4.3. Formulação do Problema

Consideramos um grafo não direcionado G(S, E), sendo ”S”o conjunto de vértices (switches), e ”E”as arestas que interligam os *switches*. O custo do menor caminho entre os *switches* i e j é expresso em termos de números de saltos. Assumimos que não temos um conjunto de implantação de controlador, visto que é tratado o problema apenas com

um único controlador que, por sua vez, tem a capacidade necessária para o número de *switches*.

A capacidade do controlador é definida por $\left\langle u_1, u_2, \cdots, u_{|F|} \right\rangle$, bem como seu número máximo de requisições por segundo expresso em u_m. Assim, o número máximo permitindo entre o switch e o controlador é representado por d_{ij}. Também consideramos o *Flow Setup Time*, que é o custo de configuração de uma nova regra de fluxo, o qual é a soma dos saltos do switch até o controlador e da volta do controlador até o switch. O custo de FST é representado pela equação 1, que mostra o custo para o pedido de configuração da rota do fluxo.

$$C_p^R = \sum_{i \in S} \sum_{i \in S} \tau_{ii} x_{im} d_{im} \tag{1}$$

para maiores informações sobre a equação, ver [Bari et al. 2013].

5. Modelo Genérico para Representação de Redes SDN

Este é um modelo genérico que permite representar várias características de SDN, tais como taxa de chegada de fluxos, distribuições de probabilidade e quantidade de *switches*, sem alterar a estrutura do modelo e o percentual de *Flow Setup Time* que vai ocorrer na rede (por exemplo, 40% de chance de ocorre *Flow Setup Time*). Com isso, pode-se representar diferentes características do SDN. Portanto, para criar o cenário 1 no CD++, foi desenvolvido em Python um gerador de topologia, que premite uma maior flexibilidade na definição do tamanho da rede, onde podemos setar a quantidade de nós e links, bem como definir os modelos atômico e acoplado. Além disso, foi considerado o algoritmo de menor caminho entre os switch e controlador.

Algoritmo 1: Gerador de Topologia

```
1  G ← nx.dense_gnm_random_graph(Node, Link)
2  isolated ← True
3  initialization;
4  while isolated do
5      for i in G.nodes() : do
6          if (nx.is_isolated(G, i)) then
7              isolated ← False ;
8              Break;
9          isolated ← False ;
10     if (isolated) then
11         G ← nx.dense_gnm_random_graph(Node, Link)
```

A Figura 3 mostra um exemplo do modelo DEVS para SDN. As áreas sombreadas são destinadas apenas para auxiliar o leitor a identificar as diferentes estrutura do modelo. O modelo proposto é um modelo acoplado composto pelos modelos atômicos, que são: um nó fonte (*Generator*), três nós switch; e um nó *sink(host)*.

Cada modelo atômico tem um conjunto de portas de entrada (in_1, $\cdots$, in_n) e saída (out_1, $\cdots$, out_n) unidirecionais para realizar a comunicação com outros modelos

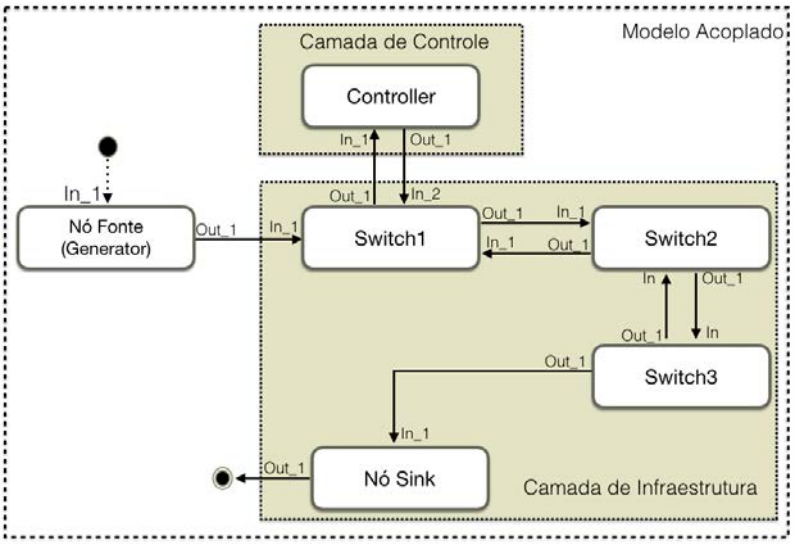


Figura 3. Topologia do Modelo DEVS de um Ambiente SDN.

atômicos. Os eventos de entrada e de saída, utilizam ambas as portas para interagir com os modelos subsequentes. Os modelos atômicos são implementados na linguagem C++ e são relacionados pelas funções herdadas da classe atômica (InitFunction, externalFunction, InternalFunction, OutputFunction), as quais são ativadas pelo método *holdIn (active, sigma)*. Maiores informações sobre funções, ver [Zeigler et al. 2000].

O nó fonte (Generator) fornece o fluxo de tráfego de dados na rede através da porta de saída para o *switch*. Ele gera a uma taxa constante, mas que é controlada pelo gerador de eventos em períodos pré-determinados. O tempo entre chegadas é simulado usando taxa de distribuição de probabilidade e, para determinar o intervalo entre chegadas de pacotes na rede, foi implementado um gerador em Python para criar os períodos de entrada. O algoritmo 2 simboliza que temos 4 períodos definidos pelos números de linhas (2, 3 e 4), e as colunas representam os seguintes detalhes de configuração: horas, minuto, segundo, milissegundo, intervalo.

Algoritmo 2: Gerador de Tráfego em Períodos de Eventos Específicos

```
1  Period ← [
2  [0, 0, 0, 100, 50],
3  [0, 0, 0, 200, 50],
4  [0, 0, 0, 300, 15]],
5  for i to range(len(period)) : do
6      #Finalsimulation
7      endHour ← period[i][0] ;
8      endMinuts ← period[i][1] ;
9      endSeconds ← period[i][2] ;
10     endMiliseconds ← period[i][3] ;
11     incrementMiliseconds ← period[i][4]
       File.write(format(period + str(random.randint(0, 20))
```

O componente *switches* consiste de duas portas de entrada e duas portas de saída. O $switch_1$, nesse exemplo, é responsável pelo acesso direto ao controlador, a partir do momento em que ele obtém as regras de tráfego, as mesmas são distribuídas para os outros *switches*. Para refinar o modelo e evitar redundância de componentes (Host) na topologia da rede, foram implementadas em cada *switch* funções de probabilidade de ocorrência de solicitação de acesso à rede. A solicitação representa o momento em que o *switch* ativa a função Host, seguidamente a solicitação, considerando os parâmetros configurados, chama a função *switch* que, por sua vez, verifica se tem a rota solicitada em sua tabela de endereçamento. Se tiver, ele encaminha o pacote para o destino (Nó Sink). Caso contrário, ele calcula a distância de acesso até o controlador para obtenção da nova rota. Este cálculo caracteriza o *flow setup time*.

O componente controlador recebe o tráfego do $switch_1$ e aguarda o período de tempo definido no *preparation time*, que representa o tempo de processamento de cálculo da rota. Seguidamente, envia a mensagem para todos os *switches*, contendo informação da rota, que é entregue inicialmente pela porta Out_1 para porta In_2 do $switch_1$, que subsequentemente repassa a mensagem. Vale ressaltar que as mudanças de estado de um componente e, consequentemente do modelo, são dependentes da ocorrência de eventos. Pois é por meio da especificação dos eventos que o projetista modela o comportamento de um modelo de simulação em DEVS.

O componente Nó Sink (Host) tem uma porta de entrada e uma porta de saída. Ele recebe os pacotes a partir do switch 3, e verifica se tem o endereço do pacote. Seguidamente, espera um período de tempo t_a. O modelo permanece no seu estado atual. Por um período de tempo definido por *Preparation Time*. Quando o tempo da função expirar, a função de saída é invocada. A função de saída envia os eventos para porta Out_1.

Todos componentes têm a seguinte ordem de processamento interno: quando a função de saída é execução, a função de transição interna será chamada para determinar o novo estado do modelo. A função de período de tempo é chamada antes de cada execução do funcionamento interno e externo, uma vez que cada estado deve ser associado a um valor de tempo único. Um outro ponto importante do modelo genérico DEVS foi a implementação do algoritmo 3, o qual realiza a avaliação de *flow setup time*, considerando o *Flow Count*. O mesmo procedimento pode ser utilizado para outras análises, como novos comportamentos que venham a ser representado no modelo.

6. Avaliação de Desempenho

Para avaliar o modelo proposto, é definido um simples estudo de caso para análise de desempenho, considerando a métrica número de fluxo - *flow count* e *flow setup time* do ambiente SDN. O principal objetivo é a validação do modelo genérico DEVS. O cenário de avaliação de SDN inclui um único controlador para toda a rede.

6.1. Geração de Topologia

Construímos nossa proposta baseada na estratégia de um único controlador que é usado para toda a rede. Esta estrutura de geração de topologias foi implementada com NetworkX [Net], mas é possível obter topologia de outras fontes.

Assim, nossos experimentos são baseados na topologia ISP com 108 nós e 306 links. A topologia referente foi definida e construída por meio de um *framework* de-

Algoritmo 3: Avaliação de *Flow Setup Time* - (AFST)

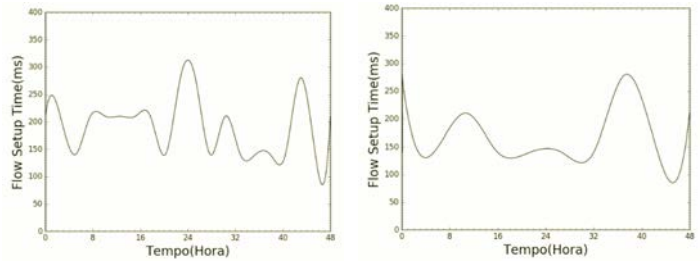
```
1  time ← [ ]
2  flow ← [ ]
3  dx ← value
4  for i to range(value) : do
5      count ← 0
6      File ← open(flowSetupTimeData" + str(i),"rb")
7      read ← cvs.read(File)
8      for row in read : do
9          if countdivdx = 0 then
10             time ← time + [str2time(row[0])] ;
11             time ← flow + [str2ms(row[2])];
12     count ← count + 1 ;
13 File.close( )
```

senvolvido em Python, que engloba várias partes de criação do modelo. Uma dessas partes foi a criação da topologia da rede, que foi baseada na biblioteca *NetworkX* [Schult and Swart 2008] e no algoritmo de menor caminho para definir o ponto exata do controlador, para auxiliar na relação de distância (saltos) exatos entre o *switch* e controlador. Esta biblioteca foi escolhida por ser implementada em Python e de fácil criação de manipulação de grafos, além de fornecer ferramentas úteis para manipular a topologia.

A comunicação entre todos os componentes permite transferir e sincronizar o status do *switch* e status de nível de porta. O controlador faz cálculo do caminho para o novo fluxo de entrada, a partir das informações que tem sobre a rede na sua base de dados local e define o caminho até o nó destino.

6.2. Resultados

Os resultados a seguir consideram um ambiente com um controlador para toda a rede. Assim, coletamos as métricas *Flow Setup Time*. Além disso, usamos interpolação para suavizar os dados. Na interpolação, é definido o tipo como *Cubic*. A Figura 4 ilustra um controlador ativo apenas na rede. O percentual permite obter uma melhor compreensão sobre o impacto da quantidade de configurações de rotas no controlador. Para os dois cenários, a simulação foi executada em um período de 48 horas com um fluxo de tráfego variando entre (0∼20(k)), mas com uma diferença por haver percentual diferente de *Flow Setup Time*. A Figura 4(a) mostra o resultado influenciado pela variação de tráfego na rede. A análise foi feita utilizando o percentual de *Flow Setup Time* igual a 50%. Já na figura 4(b) com a mesma variação de tráfego, podemos observar uma diferença na quantidade de configurações de rotas, devido ao percentual de ocorrência de *Flow Setup Time* ser 30%. Com isso, a alta variação de tráfego pode levar a ultrapassar os limites aceitáveis por um único controlador na rede.



(a) (b)

Figura 4. Flow Setup Time

7. Conclusão

Este trabalho concebeu um modelo genérico de SDN, baseado no formalismo DEVS. A partir do mesmo, é possível avaliar diversos tipos de experimentos e permite a comparação entre diferentes abordagens de gerenciamento de fluxos. Além disso, o modelo genérico proporciona uma diminuição do tempo de aprendizado para realização de testes por não necessitar do desenvolvimento, além de fornecer abstrações e ausência de protocolos e camadas de rede de baixo nível. Estes resultados mostram que as medições são precisas e escaláveis para grandes volumes de tráfego. A nossa ferramenta fornece robustez sobre a ferramenta *fs*-SDN [Gupta et al. 2013], pois também é concebido gerador de Tráfego, gerador de topologia, analisador de métricas, modelos genérico DEVS para SDN integrado ao CD++. Os experimentos são facilmente replicáveis, além disso, só é necessário utilizar uma única máquina para simular uma rede de maior escala. Um outro ponto importante, é que o trabalho é genérico o suficiente para tratar topologia com um único controlador, mas pretendemos ampliar em trabalhos futuros o desenvolvimento para múltiplos controladores, bem como adicionar meta-heurísticas.

Referências

Networkx python package website. https://networkx.lanl.gov/wiki. Accessado: 2016-01-14.

Bari, M. F., Roy, A. R., Chowdhury, S. R., Zhang, Q., Zhani, M. F., Ahmed, R., and Boutaba, R. (2013). Dynamic controller provisioning in software defined networks. In *Network and Service Management (CNSM), 2013 9th International Conference on*, pages 18–25. IEEE.

Bhuvan, V., Basil, A., Veryx, T., Mark, T., Hewlett-Packard, Manral, V., Sec, N., Banks, S., and Monitoring, V. (2016a). Benchmarking methodology for sdn controller performance draft-ietf-bmwg-sdn-controller-benchmark-meth-01. https://tools.ietf.org/html/draft-ietf-bmwg-sdn-controller-benchmark-meth-01.

Bhuvan, V., Basil, A., Veryx, T., Mark, T., Hewlett-Packard, Manral, V., Sec, N., Banks, S., and Monitoring, V. (2016b). Terminology for benchmarking sdn controller performance draft-ietf-bmwg-sdn-controller-benchmark-term-01. https://tools.ietf.org/html/draft-ietf-bmwg-sdn-controller-benchmark-term-01.

Curtis, A. R., Mogul, J. C., Tourrilhes, J., Yalagandula, P., Sharma, P., and Banerjee, S. (2011). Devoflow: scaling flow management for high-performance networks. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 254–265. ACM.

Feamster, N., Rexford, J., and Zegura, E. (2013). The road to sdn. *Queue*, 11(12):20.

Fishwick, P. A. (1995). *Simulation Model Design and Execution: Building Digital Worlds*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition.

Gupta, M., Sommers, J., and Barford, P. (2013). Fast, accurate simulation for sdn prototyping. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 31–36. ACM.

Handigol, N., Heller, B., Jeyakumar, V., Lantz, B., and McKeown, N. (2012). Reproducible network experiments using container-based emulation. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 253–264. ACM.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74.

Mello, B. A. and Wainer, G. A. (2016). Scheduling predictability in i-devs by schedulability analysis. Spring Simulation Multi-Conference / Symposium on Theory of Modeling and Simulation (TMS/DEVS), pages 622–629. SCS/ACM.

Molloy, M. K. (1982). Performance analysis using stochastic petri nets. *Computers, IEEE Transactions on*, 100(9):913–917.

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580.

Puterman, M. L. (2014). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

R. Gu, E., Li, C., and Wang, R. (2016). Problem statement of sdn and nfv co-deployment in cloud datacenters draft-gu-sdnrg-problem-statement-of-sdn-nfv-in-dc-01. https://tools.ietf.org/html/draft-gu-sdnrg-problem-statement-of-sdn-nfv-in-dc-01.

Rexford, J. and Dovrolis, C. (2010). Future internet architecture: Clean-slate versus evolutionary research. *Commun. ACM*, 53(9):36–40.

Schult, D. A. and Swart, P. (2008). Exploring network structure, dynamics, and function using networkx. In *Proceedings of the 7th Python in Science Conferences (SciPy 2008)*, volume 2008, pages 11–16.

Valmari, A. (1998). The state explosion problem. In *Lectures on Petri nets I: Basic models*, pages 429–528. Springer.

Wainer, G. (2002). Cd++: a toolkit to develop devs models. *Software: Practice and Experience*, 32(13):1261–1306.

Wainer, G. A. (2009). *Discrete-event modeling and simulation: a practitioner's approach*. CRC Press.

Yu, M., Rexford, J., Freedman, M. J., and Wang, J. (2011). Scalable flow-based networking with difane. *ACM SIGCOMM Computer Communication Review*, 41(4):351–362.

Zeigler, B. P., Praehofer, H., and Kim, T. G. (2000). *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems*. Academic press.