Modelos Semânticos em Bancos de Dados Baseados em Grafos para Aplicações de Controle de Redes Definidas por Software

Talita de Paula Cypriano de Souza[1], Luciano Bernardes de Paula[2], Mateus Augusto Silva Santos[1], Christian Esteve Rothenberg[1]

[1]Faculdade de Engenharia Elétrica e de Computação (FEEC),
Universidade Estadual de Campinas (Unicamp)
13083-970, Campinas - SP - Brasil

[2]Instituto Federal de São Paulo
12929-600, Bragança Paulista - SP - Brasil

cypriano@dca.fee.unicamp.br, lbernardes@ifsp.edu.br

msantos@dca.fee.unicamp.br, chesteve@dca.fee.unicamp.br

Abstract. *At the core of any network control and management system is the rich representation and maintenance of network topology information models. Graph Databases are an appealing alternative to traditional relational models when data is highly interconnected and topology information is important. Using metadata to describe how data is interconnected by means of Web Semantic standards is increasingly gaining ground. This paper presents an architectural approach to maintain the network information in a graph database derived from a semantic model in the context of Software Defined Networking (SDN). To ensure that this information can be used by SDN controllers in a timely and high-performance manner, the network model (NML - Network Markup Language) is imported into a modern, Web-scale graph database technology (Neo4j). To validate our proposal, a proof of concept implementation is evaluated against a representative set of SDN application primitives.*

Resumo. *No centro de qualquer sistema de controle e gerência de uma rede de computadores está a representação detalhada e a manutenção de modelos de informação sobre sua topologia. Bancos de dados baseados em grafos são uma alternativa atraente ao tradicional modelo relacional, quando os dados são altamente conectados e informações topológicas são importantes. A utilização de metadados compatíveis com os padrões da Web Semântica para descrever como dados são interconectados vem cada vez mais ganhando espaço. Este artigo apresenta uma arquitetura que mantém as informações de rede em grafo mapeado segundo um modelo semântico no contexto de Software Defined Network (SDN). Para garantir que essas informações possam ser utilizadas por controladores SDN de maneira adequada em relação a tempo e alta performance, um modelo semântico para redes (NML - Network Markup Language) é aplicado em um moderno e escalável banco de dados baseado em grafo (Neo4j). Para validar a proposta são apresentados os resultados de uma avaliação de prova de conceito em que um grupo representativo de primitivas de aplicações SDN são testadas.*

1. Introdução

Redes definidas por Software, ou em inglês *Software-Defined Networking* (SDN) [Kreutz et al. 2015], tem se apresentado como uma abordagem inovadora para a construção e operação de redes de computadores com base na identificação e implantação de novas abstrações nos planos de controle e encaminhamento das redes. A abstração mais discutida que foi introduzida por SDN é a do conceito de fluxos de pacotes e sua implementação via um protocolo aberto como o OpenFlow [McKeown et al. 2008]. Assim, é possível oferecer *Applications Programming Interfaces*(APIs) que permitem ao controlador SDN definir (remotamente) o comportamento dos comutadores que suportam o protocolo.

Uma outra abstração para o plano de controle de redes SDN que tem recebido até o momento menos atenção é a de topologia de rede. Na sua versão mais simples, uma topologia pode ser representada como um grafo que interliga os nós da rede usando arestas em função da sua conectividade, seja lógica ou física. Grafos e topologias no geral são (e continuarão sendo) o principal pilar de qualquer abordagem de arquitetura de rede, independentemente de seguir um modelo tradicional com plano de controle totalmente distribuído ou modelos mais centralizados conforme praticado em redes SDN. Seja qual for a abordagem, grafos e suas implementações em estruturas de dados são itens fundamentais para as aplicações de controle que implementam as funções lógicas da rede (e.g. geração de árvores mínimas, cálculo de melhores caminhos, caminho de recuperação).

Este trabalho foca precisamente no problema de suportar modelos semânticos para definir a abstração de uma rede e sua implementação eficiente em uma base de dados orientada a grafos. O trabalho combina resultados recentes na área de padrões relacionados com Web Semântica para descrição de computadores (NML - *Network Markup Language*) com tecnologias modernas de bases de dados orientadas a grafos (Neo4j) para sua aplicação no contexto de redes SDN. Os modelos semânticos são representados como grafos e carregados na base de dados que se torna a estrutura de dados que mantém o estado da rede SDN e que oferece uma série de primitivas para facilitar a programação das aplicações de controle SDN. Além de facilitar um desenvolvimento de aplicações e sua interoperabilidade com diferentes controladores (inclusive em cenários distribuídos e multi-domínio), o suporte de uma linguagem semântica para modelar redes SDN abre oportunidades para adicionar raciocínio lógico e técnicas de verificação formal.

Para a validação prática em uma implementação de prova de conceito foram geradas topologias de redes com tamanhos variados (até 10.000 nós) e foram então analisadas e testadas consultas referentes ao contexto de uma aplicação de controle SDN, verificando quais são atendidas tanto pelo modelo semântico quanto pelas primitivas de consultas do banco baseado em grafos. Em resumo, contribuições deste artigo são:

1. Aplicação de notação semântica (NML) para redes multi-camada no contexto de controladores SDN com interfaces a bancos de dados baseado em grafo (Neo4j);
2. Mapeamento de primitivas de uma aplicação SDN encontradas na literatura (NetGraph [Raghavendra et al. 2012]) em consultas / APIs do banco de dados;
3. Identificação de limitações da linguagem semântica NML no suporte de primitivas de aplicação controle SDN;
4. Avaliação experimental do desempenho do sistema proposto em protótipo.

O artigo está organizado da seguinte forma: a Seção 2 apresenta a fundamentação teórica e trabalhos relacionados; na Seção 3, a arquitetura proposta é detalhada assim como as primitivas utilizadas; na Seção 4 é descrito o cenário de testes e os resultados são analisados; a Seção 5 conclui e apresenta os trabalhos futuros.

2. Fundamentação Teórica e Trabalhos Relacionados

Esta seção apresenta os fundamentos teóricos e trabalhos relacionados, começando pela linguagem de marcação NML (*Network Markup Language*) para a descrição de uma rede, bancos de dados orientadas a grafos e aplicações de controle SDN.

2.1. NML (*Network Markup Language*)

A utilização dos padrões relacionados com a Web Semântica tem aumentado atualmente. O emprego de OWL (*Web Ontology Language*) e RDF (*Resource Description Language*) na criação de metadados possui vantagens em relação a outras opções como o *XML schema* ou UML. Esses padrões foram concebidos com o intuito de criar a base para que máquinas pudessem compreender dados. Basicamente, um metadado em RDF/OWL descreve algo em triplas na forma sujeito, predicado, objeto. Um objeto de uma tripla pode ser o sujeito de outra, e assim os metadados podem ser relacionados entre si.

Um exemplo de linguagem de marcação que utiliza os padrões RDF/OWL para descrever redes de computadores é o NML (*Network Markup Language*) [van der Ham et al. 2013b]. Trata-se de uma linguagem que descreve os elementos em uma rede do ponto de vista das suas interconexões e elementos interconectados. O NML derivou do NDL (*Network Description Language*), criado nos anos 2000.

O NML (*Network Markup Language*) tem como objetivo descrever redes multi-camadas e multi-domínios. A especificação dessa linguagem de marcação define que uma rede multi-camadas pode ser uma rede virtualizada ou mesmo uma rede utilizando diferentes tecnologias. Com o NML é possível descrever uma topologia de rede, suas capacidades em termos de serviços e sua configuração. NML tem foco em topologias orientadas à conexão, ou seja, aquelas nas quais o encaminhamento é feito baseado em um fluxo com label, tais como uma VLAN, um comprimento de onda ou um slot de tempo. Esse modelo também pode ser utilizada para descrever redes físicas ou orientadas à pacotes, entretanto, o seu atual esquema base não contém classes ou propriedades para tratar atributos como degradação de sinal ou tabelas de roteamento [van der Ham et al. 2013b].

No trabalho de van der Ham et al. [van der Ham et al. 2013b] é feita uma revisão de alguns padrões encontrados na literatura para descrição formal de topologias de redes de computadores. Essa revisão conclui que há pouca pesquisa sobre o assunto. As existentes geralmente são voltadas para *grids* e *cloud computing*. Nenhum padrão é amplamente adotado, dessa forma a escolha de um formato depende das necessidades do cenário. Devido a suas caraterísticas inovadoras e utilização em recentes projetos de redes (e.g. [NOVI 2010][Escalona et al. 2011][Grosso et al. 2011]), o presente trabalho optou pelo NML em sua modelagem.

Nos trabahos de van der Ham et al. [van der Ham et al. 2013a] e de Aertsen [Aertsen 2014] é possível ver os diferentes elementos definidos no *schema* do NML. O mais importante é o *Network Object*, sendo que um *Node*, uma *Port* e um *Link* são

especializações do primeiro. A parte configurável de um *Network Object* pode ser modelado como um *Service*, por exemplo, o serviço de encaminhamento entre portas em um dispositivo. Outro exemplo de serviço que pode ser citado é o de adaptação, que seria o cruzamento entre diferentes tecnologias. O encapsulamento de um pacote IP em um quadro *Ethernet* é um exemplo de adaptação.

Uma das grandes vantagens do NML é a possibilidade de se estender o modelo de acordo com a necessidade. Por exemplo, o CineGrid [Grosso et al. 2011] é uma comunidade multidisciplinar que explora os avanços das infraestruturas de rede e as adapta para o cinema digital. Esse projeto opera como um *testbed* distribuído por vários continentes. Para gerenciar o projeto, a comunidade do CineGrid utiliza a CDL (*CineGrid Description Language*) [van der Ham et al. 2011] na descrição de sua infraestrutura. O CDL deriva diretamente do NML, importando classes e implementando extensões quando necessário.

O projeto NOVI [NOVI 2010][Soundararajan and Kakaraddi 2014] tem como objetivo federar plataformas para a Internet do futuro. Um dos desafios do projeto é prover interação entre diferentes plataformas. Tanto as requisições quanto os serviços de monitoramento requerem que os diferentes recursos estejam descritos a partir de um modelo. No projeto NOVI, é utilizado uma ontologia genérica baseada no INDL (*Infrastructure and Network Description Language*) que deriva do NML.

O projeto GEYSERS [Escalona et al. 2011] tem como uma de suas inovações a virtualização de infraestruturas ópticas. O GEYSERS *Information Modeling Network* (IMF), baseado no INDL/NML provê um modelo para a descrição de dispositivos de redes ópticas, como por exemplo *switches* ópticos. Esses modelos complexos foram feitos a partir de extensões no INDL/NML, fato que mostra a sua versatilidade.

2.2. Banco de Dados baseado em Grafo

O tradicional modelo de base de dados relacional (*Relational Database Model* - RDBM) é consolidado, consistente e suas vantagens e desvantagens são bem conhecidas [Miller 2013]. Entretanto, em algumas tarefas, nas quais a informação topológica e a interconectividade dos dados são importantes, esse modelo apresenta limitações quando comparado com outras abordagens. Nesses casos, a manipulação de dados em um banco relacional pode ser mais complexa e consumir mais tempo.

Nesse contexto, uma nova categoria de modelo de banco de dados surgiu, chamada *Not Only SQL* (NoSQL). Algumas de suas vantagens são escalabilidade, escalonamento horizontal e esquema livre [NOSQL 2015]. Nos bancos de dados baseados em grafos (*Graph Database* - GDB), os quais pertencem à categoria NoSQL, os dados são armazenados como um grafo, composto por vértices e arestas que representam dados e suas relações. Esse cenário pode ser representado como entidades (vértices) e como essas se relacionam através de suas relações (arestas) [Robinson et al. 2013].

Essa abordagem permite a modelagem mais natural de diversos tipos de cenários em diferentes domínios como a Web Semântica, redes de computadores, motores de recomendação, entre outros. Devido ao crescimento desses domínios, várias soluções têm sido propostas, cada uma delas com suas próprias características e funcionalidades. Mais detalhes podem ser encontrados no trabalho de Jouili e Vansteenberghe [Jouili and Vansteenberghe 2013], no qual os autores apresentam a comparação entre importantes implementações desse tipo de banco de dados.

No trabalho de Robinson et al. [Robinson et al. 2013] os autores destacam duas características acerca dos modelos de GDB, são elas: “Armazenamento Nativo de Grafo”e “Processamento Nativo de Grafo”. Eles ressaltam que alguns modelos não possuem armazenamento nativo de grafo, ou seja, serializam o grafo em um modelo relacional, orientado a objeto ou outra proposta. E o processamento nativo requer que o cada elemento possua um apontador para o elemento adjacente e não da indexação de cada elemento.

No trabalho de Soundararajan e Kakaraddi [Soundararajan and Kakaraddi 2014] os autores utilizam um banco de dados baseado em grafo para tarefas de auditoria em *cloud*. Eles implementam consultas na linguagem *Cypher* para solucionar essas tarefas, linguagem que se mostrou intuitiva e extensível. Entre as consultas criadas estão: análise de risco, para determinar as VMs que seriam afetadas de uma rede e *datastore* em caso de uma queda de energia; reporte simples, para verificar qual é o arranjo de armazenamento que estão sendo usados pelas VMs e comparação de inventário, para verificar se duas hierarquias são equivalentes, entre outras.

Grafo de propriedades (*Property Graph*) é o modelo de grafo mais comum, o qual é um grafo direcionado, com rótulos nas arestas e utiliza atributos [Jouili and Vansteenberghe 2013]. Em um grafo de propriedades, os nós contém propriedades (pares de chave-valor) e as relações são nomeadas, direcionadas (tendo um nó inicial e um final) e também podem conter propriedades [Jouili and Vansteenberghe 2013].

O Neo4j [Miller 2013] é um GDB desenvolvido pela Neotechnology e possui versão *open-source* e versões comerciais. Algumas de suas características são: suporte a transações ACID, alta disponibilidade e alta velocidade em consultas. O modelo de grafo que o Neo4j utiliza é o grafo de propriedades. No trabalho Jouili e Vansteenberghe [Jouili and Vansteenberghe 2013] é apresentada uma comparação entre diferentes implementações de GBD, na qual o Neo4j se destaca dentre os GBDs testados. Além disso, o Neo4j é caracterizado com armazenamento e processamento de grafo nativo [Robinson et al. 2013].

Em um RDBM, para recuperar dados com grande interconexão são necessárias operações complexas do tipo *join*. Um GDB é feito para resolver esse tipo de problema e apresentar resultados com alto rendimento [Holzschuher and Peinl 2013]. O tipo de linguagem de consulta para obter dados de um GDB é chamado *traversal*. Isso significa que a consulta “percorre”o grafo através dos nós e suas arestas. Em [Holzschuher and Peinl 2013], os autores apresentam uma comparação entre as linguagens de consulta disponíveis para o Neo4j. As consultas podem ser feitas por meio da linguagem *Cypher*, linguagem declarativa similar ao SQL. Para a implementação prática apresentada nesse trabalho, foi escolhida a linguagem *Cypher*. Ela foi projetada para ser de fácil leitura e entendimento dos desenvolvedores, além de permitir escrever consultas que busque no GDB dados que combinem com determinado padrão [Robinson et al. 2013], característica que permitiu aplicar diferentes padrões de acordo com cada primitiva. Por exemplo, considere a contagem de conexões de saída em um nó A:

```
1. MATCH (n:Node)-[:hasOutboundPort]->(p:Port)-
[:isSource]->(l:Link)
2. WHERE n.name="A"
3. RETURN COUNT(l) AS CountOutDegree
```

Nessa consulta, *Cypher* percorre os nós e relacionamentos buscando a combinação com o padrão definido, em seguida faz a contagem dos *Links* que encontrados.

2.3. Aplicações de controle SDN

Em relação ao uso de grafos aplicados em um contexto de SDN, o controlador Onix [Koponen et al. 2010] pode ser considerado um trabalho seminal. Onix é uma plataforma de controle desenvolvida por pioneiros na tecnologia OpenFlow e implementa um plano de controle para redes SDN como um problema de sistemas distribuídos se apoiando em dois tipos de bases de dados em função dos requisitos de consistência do estado das aplicações. Onix utiliza grafos para agregar informações de mais baixo nível e distribuir o problema de manutenação das informações entre diferentes controladores. Onix ainda oferece APIs aos desenvolvedores de aplicações de controle, o que inclui uma visão centralizada do estado da rede que simplifica e abstrai detalhes de infraestrutura física.

O uso de grafos em SDNs também é considerado no trabalho de Pantuza et al. [Pantuza et al. 2014] para permitir que módulos de um controlador possam obter informações sobre a topologia da rede. O artigo ainda apresenta experimentos que mostram o suporte à representação dinâmica da rede. Em especial, os autores implementaram uma árvore geradora de custo mínimo que é mantida em tempo real sobre o grafo da rede.

O trabalho supracitado é similar ao NetGraph [Raghavendra et al. 2012], pois além de suportar atualizações periódicas do estado da rede, a biblioteca NetGraph também oferece resultados de consultas que podem ser utilizados por um controlador SDN. Uma característica peculiar do NetGraph é o pré-cálculo de determinadas operações para otimizar o tempo de consultas. Por exemplo, menores caminhos entre pares de nós da rede são pré-calculados de forma parcial, o que pode ser utilizado por algoritmos de roteamento.

Entretanto, essas propostas não fazem uso de uma notação semântica para a modelagem dos dados para gerar o grafo de representação da rede e não consideram o armazenamento desse grafo de forma persistente em banco de dados, principais aspectos explorados para a aproposta deste artigo.

3. Proposta de Arquitetura

O principal requisito do projeto de arquitetura é permitir que controladores SDN suportem um modelo semântico, representado como um grafo instancia do em um banco de dados escalável e de alto desempenho. Além disso, o banco de dados deve ser facilmente integrado com o controlador SDN que oferece primitivas a aplicações de controle via interfaces *Northbound* (e.g. REST) e manter o estado da rede SDN a partir da comunicação com *switches* pelas interfaces *Southbound* (e.g. OpenFlow, NETCONF).

Vislumbra-se ainda a comunicação entre diferentes controladores a partir de uma interface *east-west*, permitindo receber ou transmitir descrições da rede com o uso de um modelo semântico. Esse método abre novas oportunidades de operação de redes SDN, já que um controlador poderia mapear o relacionamento entre múltiplas fontes de informação e selecionar ações apropriadas para oferecer novos serviços [Pulkkinen et al. 2012].

A arquitetura proposta neste trabalho é apresentada na Figura 1. Aplicações externas ou internas a um controlador SDN podem realizar consultas em um GDB. Isso

permite obter informações referentes a características e situação da rede de forma centralizada ao invés de cada aplicação realizar sua consulta separadamente. Esse banco, por sua vez, recebe informações atualizadas por meio de módulos que utilizam interfaces com o plano de dados ou com outros controladores.

O GDB propicia ao controlador SDN informações resultantes de primitivas do tipo: o menor caminho entre dois nós, a contagem do grau de conectividade de um determinado nó, seus vizinhos, entre outras, além da possibilidade de criação de outras primitivas combinando as já existentes (e.g. todos os caminhos via diferentes camadas entre máquinas virtuais em hosts com interfaces 10G). Essas informações ajudam o controlador SDN e suas aplicações a tomarem decisões em relação à atuação na rede de forma precisa e rápida, como ilustrado nas próximas seções.

Como mencionado anteriormente, este trabalho adota o Neo4j como escolha de implementação, justificada pelo seu desempenho em comparação com outras implementações de GDB [Jouili and Vansteenberghe 2013]. Em relação ao modelo semântico, o uso de NML permite trabalhar com representações específicas para redes de computadores sem preocupação com detalhes de infraestrutura. Por ser definido seguindo os padrões da Web Semântica, o NML é flexível a novas extensões, podendo ser adaptado conforme a necessidade.

3.1. Primitivas

O objetivo deste trabalho é mostrar como obter informações de um grafo modelado em NML e indexado em um GDB. Foram identificadas primitivas do GDB similares às consultas e outras que necessitam certa adequação para a obtenção dos resultados corretos.

No que diz respeito à implementação, primitivas relacionadas com o grau de um nó podem ser obtidas com o uso do protocolo LLDP (*Link Layer Discovery Protocol*), o que já é realizado por implementações de controladores SDN como o *OpenDaylight*. O custo de uma conexão entre dois nós pode ser definido a partir da latência ou da largura de banda utilizada. O *OpenFlow* permite obter esse tipo de métrica através de mensagens

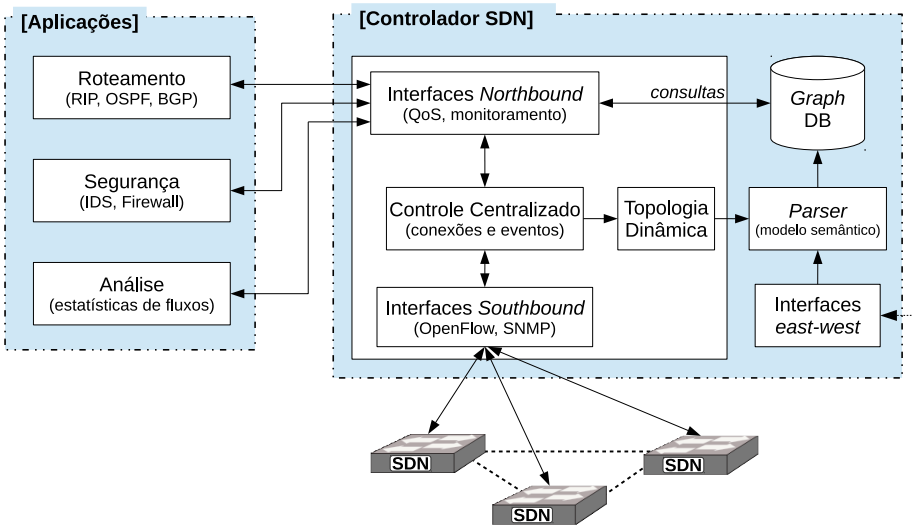

Figura 1. Arquitetura proposta integrado GBD com suporte a NML.

"*Echo request/reply*", porém há melhor precisão ao se utilizar métodos específicos, como o protocolo SNMP ou ainda o sistema sFlow.

Foi considerado que uma primitiva é suportada pelo modelo semântico quando esta pode ser respondida a partir dos atributos e relacionamentos de um grafo modelado no NML. Da mesma forma, foi considerado que uma primitiva tem suporte do GBD quando esta pode ser respondida por meio de uma ou mais consultas na linguagem *Cypher*.

As primitivas `setEdgeWeight` e `getEdgeWeight`, respectivamente, atribui custo para determinada conexão entre dois *Nodes* e obtém esse custo. Essas primitivas não são suportadas pelo modelo semântico, pois o *schema* do NML não possui atributo de custo de conexão. Esse problema poderia ser resolvido com uma extensão do modelo, na qual seria acrescentado um atributo de custo à uma entidade *Link*. O Neo4j por sua vez dá suporte às duas primitivas, pois o modelo de grafo utilizado pelo banco, grafo de propriedades, permite atributos tanto nos nós quanto nas relações.

Assim, essas primitivas poderiam ser testadas de duas formas: (*i*) criando um atributo para um relacionamento, no qual essa modelagem seria possível acrescentá-lo nos relacionamentos *isSink* e/ou *isSource* e (*ii*) criando um atributo para o nó de tipo *Link*. Para os experimentos foi implementada a segunda solução, que respeita o modelo NML.

As primitivas `countInDegree`, `countOutDegree` e `countNeighbors` que calculam a quantidade de conexões de entrada e de saída de um *Node* e seus vizinhos, são suportadas tanto pelo modelo semântico quanto pelo banco de dados. Nos experimentos, para a contagem do grau de um *Node*, foi calculada a quantidade de *Links* ligados às portas (*Port*) de entrada e de saída de um nó e para a contagem de vizinhos, é a mesma implementação, considerando os *Nodes* conectados.

As primitivas `computeMST`, que gera a *Minimum Spanning Tree* a partir de uma origem, e a `computeSSSP` (*Single Source Shortest Path*) que retorna o menor caminho de um *Node* para todos os outros, utilizaram o mesmo recurso da linguagem *Cypher* chamado *shortestPath*. Esse recurso foi usado também para encontrar o menor caminho de cada par de *Nodes* da primitiva `computeAPSP` (*All Pair Shortest Path*). Outro recurso nativo do *Cypher* utilizado foi o `allShortestPath` para a primitiva `computeKSSSP` (*k Single Source Shortest Path*), que encontra k menores caminhos entre dois *Nodes*.

Para essas primitivas de menores caminhos foi constatado que o modelo semântico possui suporte, visto que sua modelagem gera um grafo que naturalmente permite essas operações. Nesse sentido, a primitiva `doesRouteExist` que verifica a existência de rota entre dois *Nodes* é suportada pelo modelo semântico e pelo banco de dados.

As primitivas `insert` e `delete` são suportadas pelo banco de dados e pelo modelo semântico. A inserção de um *Node* no banco de dados realiza a inserção de duas *Ports* e os relacionamentos entre eles. Da mesma forma, uma exclusão remove *Ports* e *Links* que o *Node* está relacionado, bem como seus relacionamentos.

No trabalho de Raghavendra et al. [Raghavendra et al. 2012], a biblioteca NetGraph, implementa duas principais funcionalidades: (1) consultar a topologia de rede incluindo nós e estado de links para manter um grafo de rede atualizado e (2) computar consultas do grafo e retornar os resultados da consulta de uma forma que possa ser utilizada por outros módulos para prover virtualização de redes (NaaS *Network as a Service*).

Essas consultas são apresentadas na Tabela 3.1. Na mesma tabela é apresentado se o modelo semântico (NML) utilizado para a modelagem do grafo proporciona a execução de tal primitiva sem a necessidade de alterações e se há primitiva nativa para tal consulta no GDB (Neo4J) utilizado nos testes. A última coluna se refere ao tipo da primitiva, escrita ou leitura.

Primitiva	Modelo Semântico	GDB	Leitura / Escrita
`setEdgeWeight`	Não	Sim	E
`getEdgeWeight`	Não	Sim	L
`countInDegree`	Sim	Sim	L
`countOutDegree`	Sim	Sim	L
`countNeighbors`	Sim	Sim	L
`ComputeMST`	Sim	Sim	L
`computeAPSP`	Sim	Sim	L
`computeSSSP`	Sim	Sim	L
`doesRouteExist`	Sim	Sim	L
`computeKSSSP`	Sim	Sim	L
`delete`	Sim	Sim	E
`insert`	Sim	Sim	E

Tabela 1. Primitivas da literatura

4. Avaliação Experimental e Análise de Resultados

Esta seção apresenta a metodologia utilizada para a realização dos experimentos de prova de conceito para validar a proposta do trabalho, começando pela apresentação do ambiente de teste, a descrição das topologias, a modelagem dos dados e por fim a análise dos resultados e a disponibilidade dos dados para reprodutibilidade dos experimentos.

4.1. Ambiente de testes

Os testes foram executados em uma máquina de processador Intel i7 de 2.4 GHz, 8 Gigabytes de memória RAM. Para a realização dos testes a versão do Neo4j utilizada foi a Comunity Edition 2.0.4, com licença GPLv3. Para a criação das topologias no banco de dados e execução de consultas, foi utilizada a API Java do Neo4j. Apesar da API oferecer métodos pré-definidos de manipulação no banco, como a modelagem dos nós são segundo o modelo NML, os métodos requerem adaptação, dessa forma, foi escolhida a criação e execução de consultas por meio da linguagem *Cypher*.

4.2. Topologias

Para o teste da aplicação foram criadas quatro topologias de tamanhos diferentes, utilizando o gerador de topologias BRITE [Medina et al. 2001]. Esse gerador possui algumas opções de modelos e vários parâmetros em cada modelo. A classe escolhida para a geração das topologias para os testes deste experimento foi a *Flat Router-Level* e o modelo *RouterWaxman*. Esse modelo gera uma topologia aleatória por meio da probabilidade Waxman para a interconexão dos nós da topologia. Outra característica desse modelo é que a criação de todos os nós no plano é feita da mesma forma e após a criação da topologia os atributos de largura de banda de todas as conexões também são definidos igualmente [Medina et al. 2001]. Para a geração das topologias deste experimento a localização dos nós no plano foi feita de forma aleatória.

O BRITE gera um arquivo com os nós da rede, seus atributos e as conexões entre eles. Para cada nó o gerador define sete atributos no total, mas para esse experimento foi utilizado apenas o atributo de identificador único (*NodeId*), que corresponde a propriedade *name* do modelo NML. O gerador define também nove atributos para as conexões, porém para os testes apresentados aqui esses atributos não foram considerados.

Para analisar o desempenho do GBD, foram criadas 4 topologias de 10 (*tiny*), 100 (*small*), 1.000 (*medium*) e 10.000 (*large*) nós gerados pelo BRITE. Para cada topologia, os nós foram criados no banco de dados conforme modelagem apresentada na Seção 4.3 resultando nos seguintes grafos: 76 nós (160 relacionamentos), 640 nós (1.760 relacionamentos), 4.978 nós (11.912 relacionamentos) e 109.932 nós (359.728 relacionamentos). É importante lembrar que os nós gerados no grafo do GBD pode ser dos tipos *Nodes*, *Ports* e *Links*.

4.3. Modelagem dos Dados

Para indexar os nós e relacionamentos gerados pelo BRITE no GDB, seguindo o modelo NML, foram geradas para cada nó duas portas (elemento *Port* definido no NML), sendo uma de entrada e uma de saída. Essas portas foram identificadas, respectivamente, com seu id_{in} e id_{out}. Para cada conexão entre dois nós foram criados dois *Links* (um para cada sentido de fluxo) que relacionam as portas de seus nós. A Figura 2 apresenta a modelagem da conexão entre os nós "9"e "0", suas *Ports*, *Links* e relacionamentos. Nas próximas subseções, as topologias geradas são detalhadas.

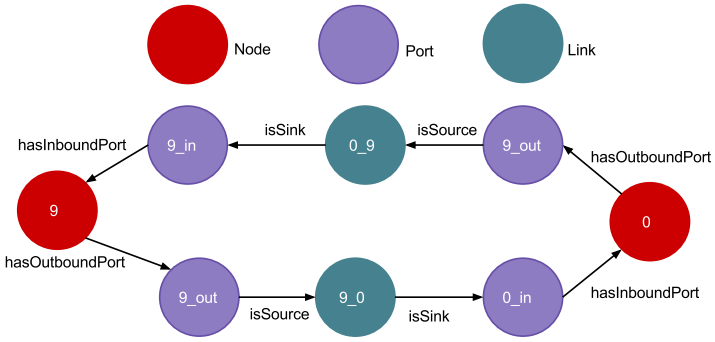

Figura 2. Visualização no Neo4j do relacionamento entre os nós "9"e "0"

4.4. Análise de Resultados

Para realizar as métricas das consultas, a aplicação desenvolvida indexou a topologia e executou, individualmente e sequencialmente, as primitivas (consultas) com parâmetros aleatórios. Durante a realização dos experimentos, foi observado que os primeiros tempos de cada primitiva foram muito superiores à média das demais consultas. Esse comportamento foi também observado no trabalho de Soundararajan e Kakaraddi [Soundararajan and Kakaraddi 2014]. A explicação para tal fato é que na primeira execução o Neo4j coloca o grafo em cache e as demais execuções são realizadas no grafo em memória, o que economiza tempo. Por esse motivo, todos os primeiros resultados foram desconsiderados e cada primitiva foi executada 1.001 vezes em cada topologia.

Nas Tabelas 2 e 3 são apresentados a média, desvio padrão, 99 percentil e 1 percentil de cada primitiva, nas topologias *small* e *large*. A coluna de 1º valor se refere ao

tempo desconsiderado para cálculo das estatísticas. Na 2 a primitiva `delete` não foi executada 1001 vezes, devido ao seu tamanho, ela foi executada 100 vezes. Essas primitivas utilizam parâmetros, que pela aplicação são gerados aleatoriamente, e a latência está relacionada com o grau de conectividade do nó utilizado como parâmetro.

As primitivas com maior latência são `countInDegree`, `countOutDegree` e `delete`, comportamento verificado em todas as topologias. Para as operações de contagem de conexões de entrada e saída de um nó, esse resultado pode ser explicado devido ao número de *hops*, ou seja, os tipos diferentes de relacionamentos que é preciso percorrer para fazer a contagem [Soundararajan and Kakaraddi 2014]. Por exemplo, para a contagem de conexões de entrada de um *Node* A, deve ser percorrido o seguinte padrão:

$$NodeA \leftarrow hasInboundPort \leftarrow Port \leftarrow isSink \leftarrow Link$$

O mesmo ocorre com a operação de exclusão, pois para essa primitiva, a consulta exclui o *Node*, suas *Ports*, *Links* conectados às portas e os relacionamentos com outras *Port*. Nesse caso, o número de *hops* é maior que nas operações de contagem de entrada e saída. Dessa forma, a latência está ligada ao nível de conectividade do nó excluído. Em seguida está a `setEdgeWeight` que no tempo geral das primitivas teve uma latência maior que as demais operações de leitura, ainda que na topologia *small* a média não tenha ficado alta. Como comparado no trabalho de Jouili e Vansteenberghe [Jouili and Vansteenberghe 2013] o Neo4j apresenta um melhor desempenho em operações de somente leitura do que em operações de leitura-escrita. É possível validar ainda mais esse comportamento, comparando com os resultados da primitiva `getEdgeWeight`.

Para as demais primitivas, a Figura 3 mostra um comparativo entre as primitivas de leitura `computeSSSP`, `computeKSSSP` e `doesRouteExist` e a primitiva de escrita `insert` para as topologias *small*, *medium*, *large* Não foram inseridas todas as primitivas, devido aos altos valores, para manter uma escala que permita visualização dos resultados. Em geral, as primitivas de menores caminhos apresentam boa performance nas quatro topologias. Um comportamento interessante observado foi que a primitiva que calcula todos os pares de menor caminho (`computeAPSP`), é inferior quando comparado com a primitiva de k menores caminhos entre dois nós (`computeKSSSP`) e a de menores caminhos a partir de um nó específico (`computeSSSP`).

É possível deduzir que o GDB otimiza as consultas no caso de todos os pares de menores caminhos pois pode já calcular o menor caminho entre nós intermediários. O que não ocorre nas primitivas que consideram um nó ou dois específicos. Ainda nessa análise, a primitiva de `doesRouteExist` apresenta maior latência, uma vez que calcula o caminho entre nós específicos. A primitiva `computeMST` possui a mesma implementação de `computeSSSP`, como é possível observar nos seus resultados próximos.

4.5. Disponibilidade dos Dados e Reproducibilidade dos Experimentos

Todos os dados e código fonte usados nos experimentos estão disponíveis em um repositório público[1]. As informações disponibilizadas incluem os modelos NML, topologias BRITE, *parsing scripts*, consultas no Neo4j e código *gnuplot* usado no artigo.

[1]https://github.com/intrig-unicamp/NML-Neo4j

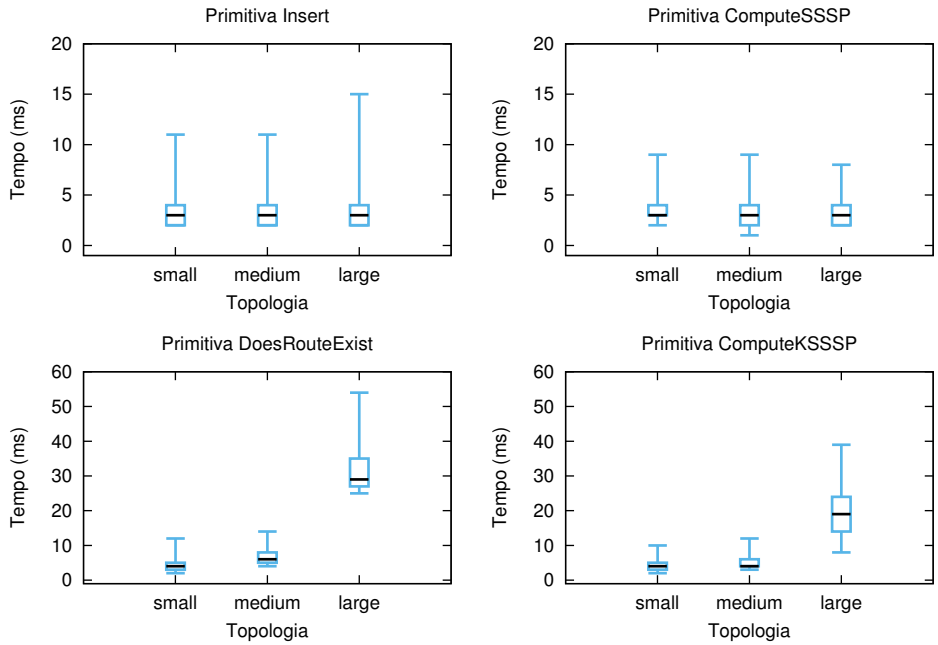

Figura 3. Tempo de resposta das primitivas. Os gráficos *candlesticks* apresentam o valor médio, os quartis, e os valores max/min como 95-percentil.

5. Conclusão e Trabalhos Futuros

Esse trabalho apresentou um estudo sobre a indexação de uma rede seguindo um modelo semântico (NML) em um banco baseado em grafo (Neo4j) no contexto de redes definidas por software (SDN). Foi proposta uma arquitetura para a integração de um banco de dados baseado em grafos com uso de um modelo semântico para instanciar o grafo de uma rede, com o objetivo de fornecer para o controlador SDN reposta a determinadas primitivas. A partir dos experimentos, o NML se mostrou compatível com as necessidades de modelagem no cenário, entretanto uma limitação foi identificada, conforme apresentada na Seção 3.1, em que o modelo não prevê características de custo para um *Link*. Apesar da limitação identificada, o NML é um modelo extensível, ou seja, permite extensões para adaptar a necessidade do domínio. O Neo4j com o seu grafo de propriedades foi compatível e armazenou segundo a modelagem (entidades, atributos e relacionamentos) do NML sem necessidade de alteração. E a linguagem de consulta *Cypher* se mostrou flexível, uma vez que permite busca de padrões conforme a modelagem. Como visto, as primitivas utilizadas por uma aplicação SDN puderam ser reproduzidas e os resultados com o tempo das consultas são promissores.

Pretende-se a continuação do trabalho em múltiplas direções: (*i*) avaliar o desempenho como cargas de trabalho dinâmicas e aplicações no controlador OpenDaylight usando REST APIs; (*ii*) desenvolvimento de novas primitivas para consumo de controladores via interfaces East-West; (*iii*) explorar otimizações na latência e capacidade do sistema via pré-calculo e uso de propriedades para (des)habilitar nós no grafo; (*iv*) desenvolvimento de extensões do modelo semântico (NML) para refletir a descrição de redes SDN e suportar a modelagem de funções de rede virtualizadas (NFV - Network Function Virtualization) oferecidas como serviço.

Primitiva	1o. Valor	Média	Desvio Padrão	99 percentil	1 percentil
countInDegree	1750,00	11,32	8,37	51,00	4,00
countOutDegree	1587,00	7,98	6,06	38,03	4,00
insert	1268,00	4,41	4,14	22,01	1,00
countNeighbors	1725,00	8,52	7,02	37,01	3,00
doesRouteExist	1179,00	5,01	3,44	21,00	2,00
computeSSSP	1101,00	4,19	3,39	19,01	1,00
computeKSSSP	1251,00	5,14	4,42	24,02	2,00
setEdgeWeight	60,00	3,97	1,98	11,00	2,00
getEdgeWeight	32,00	3,35	1,81	10,00	1,00
computeMST	1496,00	4,03	3,59	22,01	1,00
delete*	1204,00	36,78	13,28	62,96	24,24
computeAPSP	1171,00	2,46	1,77	10,01	1,00

Tabela 2. Tempos (ms) da Execução das Primitivas - Topologia *small*

Primitiva	1o. Valor	Média	Desvio Padrão	99 percentil	1 percentil
countInDegree	9713,00	653,96	91,92	1026,05	585,99
countOutDegree	6484,00	332,98	47,39	528,02	294,00
insert	5246,00	4,77	5,77	33,00	1,00
countNeighbors	4295,00	6,54	5,76	31,00	3,00
doesRouteExist	5340,00	35,24	28,54	166,43	25,00
computeSSSP	3935,00	3,49	3,08	16,00	1,00
computeKSSSP	4660,00	24,55	39,09	312,09	5,00
setEdgeWeight	421,00	218,13	57,77	448,87	180,00
getEdgeWeight	32,00	11,08	2,60	17,12	8,99
computeMST	3934,00	3,94	3,04	18,01	1,00
delete	9545,00	1026,72	134,35	1494,20	892,99
computeAPSP	4328,00	3,01	2,14	11,01	1,00

Tabela 3. Tempos (ms) da Execução das Primitivas - Topologia *large*

Referências

Aertsen, M. (2014). Verifying functional requirements in multi-layer networks: a case for formal description of computer networks.

Escalona et al. (2011). GEYSERS: A novel architecture for virtualization and co-provisioning of dynamic optical networks and IT services. In *2011 Future Network & Mobile Summit, Warsaw, Poland, June 15-17, 2011*, pages 1–8.

Grosso, P., Herr, L., Ohta, N., Hearty, P., and de Laat, C. (2011). Cinegrid: Super high definition media over optical networks. *Future Gener. Comput. Syst.*, 27(7):881–885.

Holzschuher, F. and Peinl, R. (2013). Performance of graph query languages: Comparison of cypher, gremlin and native access in neo4j. In *Proceedings of the Joint EDBT-ICDT 2013 Workshops*, EDBT '13, pages 195–204, New York, NY, USA. ACM.

Jouili, S. and Vansteenberghe, V. (2013). An empirical comparison of graph databases. In *Social Computing (SocialCom), 2013 International Conference on*, pages 708–715.

Koponen, T., Casado, M., Gude, N., Stribling, J., Poutievski, L., Zhu, M., Ramanathan, R., Iwata, Y., Inoue, H., Hama, T., et al. (2010). Onix: A distributed control platform for large-scale production networks. In *OSDI*, volume 10, pages 1–6.

Kreutz, D., Ramos, F., Esteves Verissimo, P., Esteve Rothenberg, C., Azodolmolky, S., and Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74.

Medina, A., Lakhina, A., Matta, I., and Byers, J. (2001). Brite: An approach to universal topology generation. In *Proceedings of MASCOTS 01*, page 346, Washington, DC, USA. IEEE Computer Society.

Miller, J. J. (2013). Graph database applications and concepts with neo4j. In *Proceedings of the Southern Association for Information Systems Conference*, Atlanta, USA.

NOSQL (2015). http://nosql-database.org/.

NOVI (2010). *Networking innovations Over Virtualized Infrastructures*.

Pantuza, G., Sampaio, F., Vieira, L. F., Guedes, D., and Vieira, M. A. (2014). Network management through graphs in software defined networks. In *Network and Service Management (CNSM), 2014 10th International Conference on*, pages 400–405. IEEE.

Pulkkinen, T., Sallinen, M., Son, J., Park, J.-H., and Lee, Y.-H. (2012). Home network semantic modeling and reasoning 2014. a case study. In *Information Fusion (FUSION), 2012 15th International Conference on*, pages 338–345.

Raghavendra, R., Lobo, J., and Lee, K.-W. (2012). Dynamic graph query primitives for sdn-based cloudnetwork management. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 97–102.

Robinson, I., Webber, J., and Eifrem, E. (2013). *Graph Databases*. O'Reilly Media, Inc.

Soundararajan, V. and Kakaraddi, S. (2014). Applying graph databases to cloud management: An exploration. In *Cloud Engineering (IC2E)*, pages 544–549.

van der Ham, J., Dijkstra, F., Lapacz, R., and Brown, A. (2013a). Network Markup Language Base Schema version 1. *Technical Report - Open Grid Forum*.

van der Ham, J., Dijkstra, F., Lapacz, R., and Brown, A. (2013b). The Network Markup Language (NML) A Standardized Network Topology Abstraction for Inter-domain and Cross-layer Network Applications. *TNC2013*.

van der Ham, J. J., P. Chrysa, S. J., Peter, M., Yiannos, K., Grosso, P., and Lymberopoulos, L. (2011). Challenges of an information model for federating virtualized infrastructures. *5th International DMTF Academic Alliance Workshop on Systems and Virtualization Management: Standards and the Cloud*.