Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação

Talita de Paula Cypriano de Souza

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

Monografia de Qualificação apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos exigidos no programa de Mestrado em Engenharia Elétrica. Área de concentração: Engenharia de Computação.

Orientador: Prof. Dr. Christian Rodolfo Esteve Rothenberg

Co-orientador: Prof. Dr. Luciano Bernardes de Paula

Campinas
2015

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 trabalho de mestrado tem como objetivo conceber uma arquitetura que mantém as informações de uma rede de computadores em um 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 é aplicado em um moderno e escalável banco de dados baseado em grafo (Neo4j). Para a validação da proposta serão apresentados os resultados parciais de avaliações de prova de conceito em que um grupo representativo de primitivas de aplicações SDN foram testadas.

Conteúdo

Capítulo 1

Introdução

Redes definidas por *software*, ou em inglês *Software-Defined Networking* (SDN) [1], 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. 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 [2]. Assim, é possível oferecer APIs (*Application Programming Interface*) que permitem ao controlador SDN definir (remotamente) o comportamento dos comutadores que suportam o protocolo.

Uma outra abstração importante para o plano de controle de redes SDN que tem recebido menos atenção é a da 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 (ex: geração de árvores mínimas, cálculo de menores caminhos, caminho de recuperação).

Este projeto de pesquisa foca precisamente no problema de suportar modelos semânticos para definir a abstração de uma rede e sua implementação em uma base de dados orientada a grafos. Para isso serão combinados resultados recentes na área de padrões relacionados com Web Semântica para descrição de redes de computadores (*Network Markup Language* - NML [3]) com tecnologias modernas de bases de dados orientadas a grafos (Neo4j) para sua aplicação no contexto de redes SDN.

O texto está organizado da seguinte forma: o Capítulo 2 apresenta os objetivos da pesquisa; o Capítulo 3 apresenta a fundamentação teórica e os trabalhos relacionados; Capítulo 4 apresenta a topologia; a arquitetura é detalhada assim como as primitivas utilizadas; e finalmente no Capítulo 6 o cenário dos testes parciais é descrito e os resultados analisados.

Capítulo 2

Objetivos

Neste projeto de pesquisa pretende-se representar modelos semânticos como grafos e carregá-los na base de dados. Dessa forma, a base de dados 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 o 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.

Os objetivos específicos desse trabalho 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 do modelo NML no suporte de primitivas de aplicação controle SDN;

4. Avaliação experimental do desempenho do sistema proposto em protótipo.

Capítulo 3

Revisão Bibliográfica

Neste capítulo serão apresentados os fundamentos teóricos e os trabalhos relacionados, começando pela linguagem de marcação NML (*Network Markup Language*) para descrever uma rede, passando por bancos de dados baseados em grafos e por fim aplicações de controle SDN.

3.1 *Network Markup Language* - NML

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 e assim facilitar a automatização do seu processamento. Basicamente, um metadado em RDF/OWL descreve algo em triplas na forma de sujeito, predicado e 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*) [3]. 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 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 *labels*, tais como uma VLAN, um comprimento de onda ou um *slot* de tempo. Esse modelo também pode ser utilizado para descrever redes físicas ou orientadas a pacotes, entretanto, o seu atual esquema base não contém classes ou propriedades para tratar atributos como degradação de sinal ou tabelas de roteamento [3].

No trabalho de van der Ham et al. [3] 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 grades computacionais (*grids*) e computação em nuvem (*cloud computing*). Nenhum padrão é amplamente adotado, dessa forma a escolha de um formato depende das necessidades do cenário. Devido ao fato de utilizar padrões da Web Semântica e emprego em recentes projetos relacionados a redes de computadores (e.g. [4][5][6]), o NML foi escolhido para a modelagem

neste trabalho.

Nos trabahos de van der Ham et al. [7] e de Aertsen [8] 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 de se utilizar padrões da Web Semântica, como é o caso do NML, é a possibilidade de se estender o modelo de acordo com a necessidade. Por exemplo, o CineGrid [6] 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*) [9] na descrição de sua infraestrutura. O CDL deriva diretamente do NML, importando classes e implementando extensões quando necessário.

O projeto NOVI [4][10] 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 [5] 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.

3.2 Bancos de Dados baseados 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 [11]. 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 [12]. 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) [13].

Essa abordagem permite a modelagem mais natural de diversos tipos de cenários em diferentes domínios como, por exemplo, 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 [14], no qual os autores apresentam a comparação entre importantes implementações desse tipo de banco de dados.

No trabalho de Robinson et al. [13] 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. O processamento nativo requer que cada elemento possua um apontador para o elemento adjacente e não da indexação de cada elemento.

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 [14]. Em um grafo de propriedades, os nós contém propriedades (pares de chave-valor) e as relações são nomeadas e direcionadas (tendo um nó inicial e um final), podendo também conter propriedades [14]. A Figura 3.1 apresenta um exemplo desse tipo de grafo.

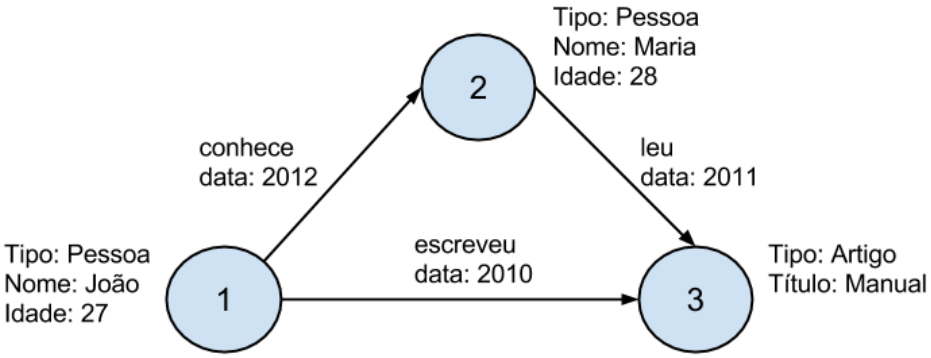

Figura 3.1: Exemplo de grafo de propriedades (adaptado de [3])

O Neo4j [11] 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 [14] é apresentada uma comparação entre diferentes implementações de GDB, na qual o Neo4j se destaca dentre os GDBs testados. Além disso, o Neo4j é caracterizado com armazenamento e processamento de grafo nativo [13].

Em um RDBM, para recuperar dados com grande interconexão são necessárias operações complexas do tipo *join*. GDBs foram planejados para resolver esse tipo de problema e apresentarem resultados com alto rendimento [15]. O tipo de linguagem de consulta para obter dados de um GDB é chamado *traversal*, pois a consulta "percorre"o grafo através dos nós e suas arestas. Em [15], os autores apresentam uma comparação entre as linguagens de consulta disponíveis para o Neo4j. As consultas podem ser feitas por meio de uma linguagem declarativa similar ao SQL chamada *Cypher*. Para a implementação prática desse projeto de pesquisa, foi utilizada a linguagem *Cypher*, pois essa 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 [13], 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, os nós e relacionamentos são percorridos buscando a combinação com o padrão definido, em seguida é feita a contagem dos *Links* que encontrados. Além de consultas

de somente leitura, a linguagem permite consultas de leitura-escrita no GDB. Por exemplo, considere a criação de relacionamento entre uma *Port* e um *Link*:

```
1. MATCH (a:Port), (b:Link)
2. WHERE a.name="A_out" AND b.name="A_B"
3. CREATE (a)-[r:isSource]->(b)
4. RETURN r
```

Nesse exemplo, primeiramente os nós (*Port* e *Link*) são consultados e em seguida é criado um relacionamento do tipo *isSource* entre eles.

Um dos trabalhos relacionados é o dos autores Soundararajan e Kakaraddi [10] que utilizam o banco de dados baseado em grafo Neo4j 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.

3.3 Aplicações de controle SDN

Em relação ao uso de grafos aplicados em um contexto de SDN, o controlador Onix [1] 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. [16] 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 [17], 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 proposta deste projeto de pesquisa.

Capítulo 4

Proposta

Este capítulo apresenta as detalhes da proposta do projeto de mestrado, primeiramente detalhando a arquitetura em seguida a análise das primitivas SDN da literatura.

4.1 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 [18].

A arquitetura proposta nesta pesquisa é apresentada na Figura 4.1. 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 [14]. 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.

4.2 Análise das Primitivas

O objetivo deste projeto de pesquisa 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 "*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

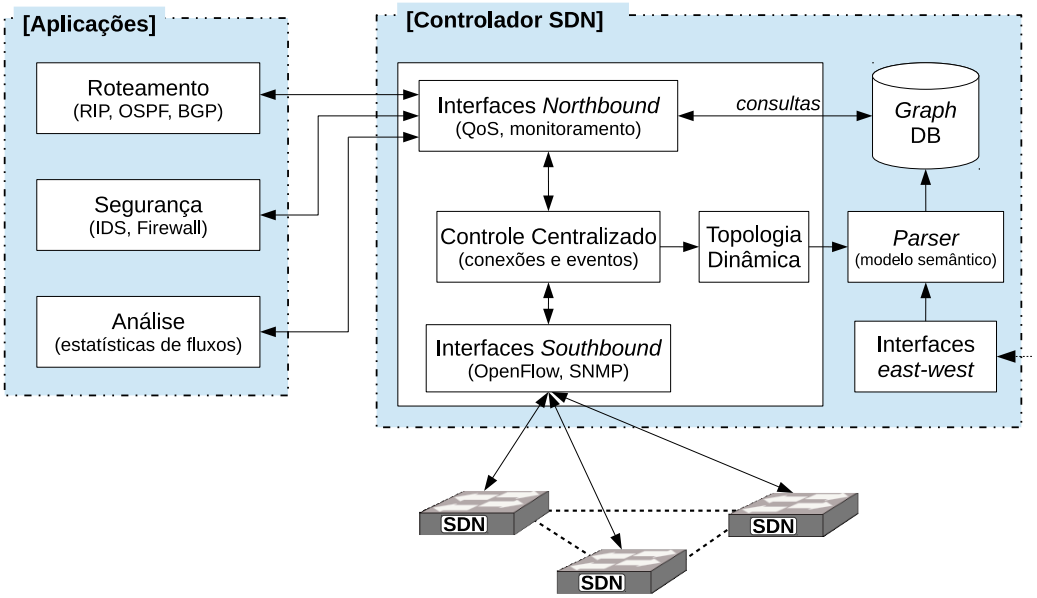

Figura 4.1: Arquitetura proposta integrado GBD com suporte a NML.

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 `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. [17], 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 **??**. 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.

Tabela 4.1: Primitivas da literatura

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

Capítulo 5

Resultados e Conclusões Parciais

Neste capítulo será apresentado uma metodologia experimental de implementação de prova de conceito e seus resultados parciais.

5.1 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.

5.2 Topologias

Para o teste da aplicação foram criadas quatro topologias de tamanhos diferentes, utilizando o gerador de topologias BRITE [19]. 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 *RouterBarabasiAlbert*. 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 [19]. 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 5.3 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*.

5.3 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 5.1 apresenta a modelagem da conexão entre os nós "9"e "0", suas *Ports*, *Links* e relacionamentos. Nas próximas subseções, as topologias geradas são detalhadas.

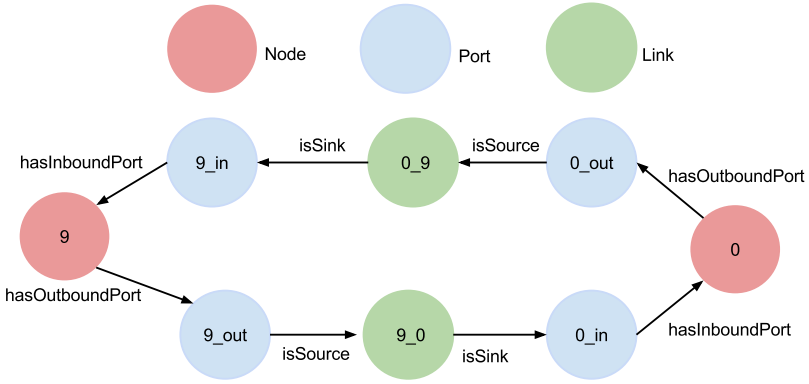

Figura 5.1: Visualização no Neo4j do relacionamento entre os nós "9"e "0"

5.4 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 [10]. 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 5.1 e 5.2 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 5.1 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 [10]. 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 [14] 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 5.2 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.

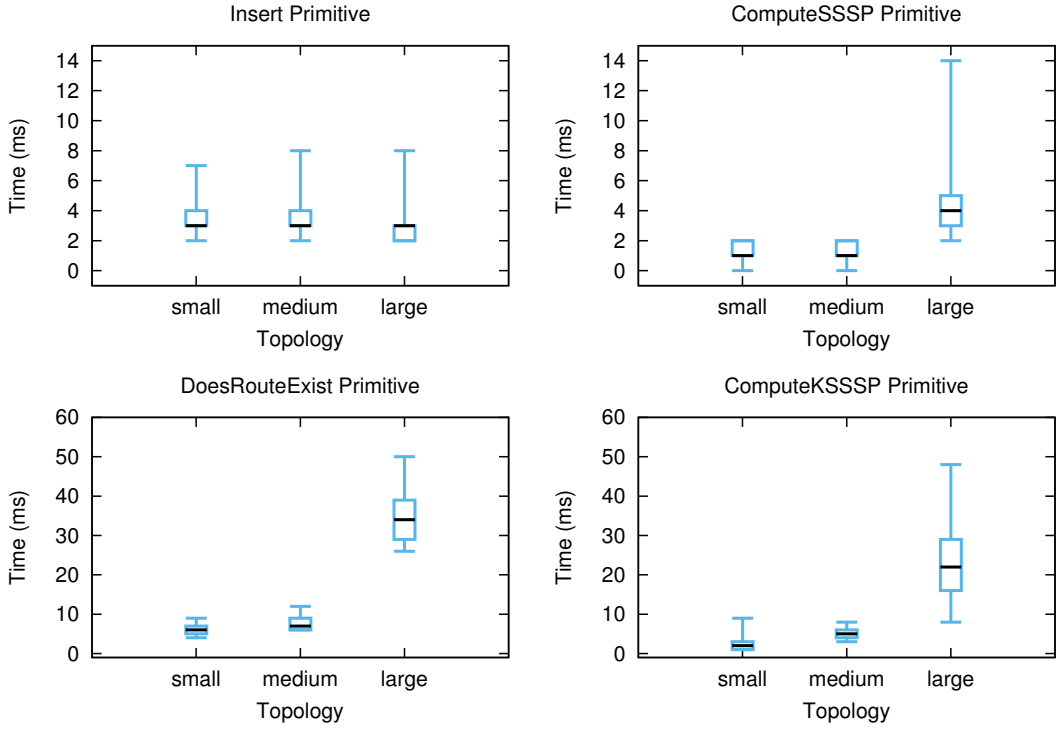

Figura 5.2: Tempo de resposta das primitivas. Os gráficos *candlesticks* apresentam o valor médio, os quartis, e os valores max/min como 95-percentil.

5.5 Comparação com um Modelo Relacional

No sentido de obter resultados de performance de um tradicional RDBM, foram executados um conjunto das primitivas SDN usando uma aplicação Java com MySQL (version 5.6.17) [20] e a linguagem de consulta SQL. A Figura 5.3 apresenta o diagrama Modelo Entidade Relacionamento (MER). O modelo segue os relacionamentos apresentados na Figura 5.1 representando o grafo como tabelas usando chaves primárias e estrangeiras. Assim, os relacionamentos são de *um-para-muitos* entre *Node* e *Port*, enquanto que um *Link* é um relacionamento recursivo *muito-para-muitos* entre duas *Ports*.

Durante o projeto da aplicação de teste foi possível observar uma vantagem prática do GDB, em que tarefas de modelagem são consideravelmente mais naturais comparadas ao RDBM, o que é esperado, considerando que é um projeto de rede, com interconexão de dados. Tabela 5.3 apresenta os tempos de execução para um subconjunto das primitivas no caso da topologia *large*.

Os resultados para as primitivas `delete` e `countInDegree` são mais rápidas com o MySQL do que com o Neo4j, um fato que pode ser esperado a partir da latência acumulativa (número de *hops*) explicado na seção anterior. Em contra partida, no caso de inserção, o tempo de execução da primitiva `insert` foi muito mais alto do que no Neo4j, provavelmente devido à necessidade de inserir dados em duas tabelas (*Node* e *Port*). Para a primitiva `computeSSSP` os resultados foram um pouco mais lentos que no Neo4j. Destaca-se aqui o fato de que para

Tabela 5.1: Tempo (ms) de execução das primitivas - Topologia *small*

Primitiva	Média	Desvio Padrão	Percentil 99
`setEdgeWeight`	8.78	3.23	23.02
`getEdgeWeight`	1.73	0.76	3.00
`countInDegree`	17.94	11.36	65.01
`countOutDegree`	8.35	3.46	23.00
`countNeighbors`	6.16	22.43	14.07
`doesRouteExist`	6.55	3.82	15.02
`computeMST`	1.12	0.66	2.00
`computeSSSP`	1.34	1.38	4.00
`computeKSSSP`	2.94	3.44	12.00
`computeAPSP`	1.04	0.84	4.01
`delete`	20.71	7.20	48.01
`insert`	3.66	3.26	15.02

Tabela 5.2: Tempo (ms) de execução das primitivas - Topologia *large*

Primitiva	Média	Desvio Padrão	Percentil 99
`setEdgeWeight`	162.33	9.46	205.01
`getEdgeWeight`	1.70	0.74	4.00
`countInDegree`	854.53	146.77	1399.05
`countOutDegree`	425.17	68.36	699.02
`countNeighbors`	4.45	2.27	10.01
`doesRouteExist`	37.51	29.09	73.06
`computeMST`	1.44	1.25	3.02
`computeSSSP`	5.47	4.98	29.00
`computeKSSSP`	26.21	37.23	81.04
`computeAPSP`	1.04	0.68	3.01
`delete`	1053.89	162.55	1637.02
`insert`	3.57	3.21	16.01

calcular as primitivas de menores caminhos uma implementação Java (baseada em Dijkstra [21]) foi necessária devido à falta de consultas do tipo de menores caminhos nativas no MySQL. Primeiramente, a implementação adaptada do algoritmo Dijkstra carrega em memória todos os nós e suas adjacências utilizando consultas do tipo `SELECT`. A média de tempo para essa tarefa de pré-execução dos dados é de aproximadamente 2 segundos. Novamente, as primitivas orientadas a grafo nativas do Neo4j podem ser vistas como vantagens do GDB sobre o RDBM em uma perspectiva de desenvolvimento de aplicação. Do mesmo modo, a primitiva `computeAPSP` apresenta desempenho comparável, mas não requer qualquer esforço de desenvolvimento extra em uma abordagem GDB devido a funções nativas do Neo4j para cálculo de todos os pares de menores caminhos.

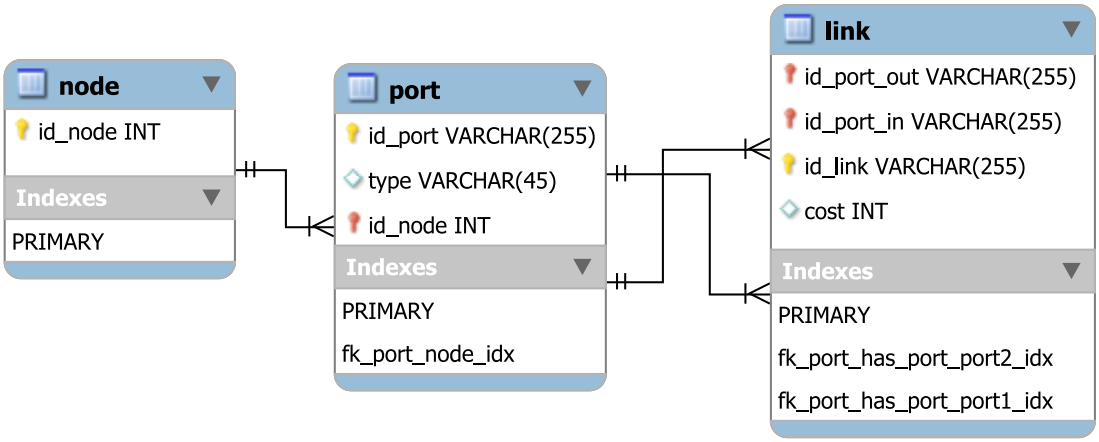

Figura 5.3: Modelo Entidade-Relacionamento (MER)

Tabela 5.3: Tempo (ms) de execução das primitivas no RDBM - Topologia *large*

Primitiva	Média	Desvio Padrão	Percentil 99
`countInDegree`	1.39	4.57	22.02
`computeSSSP`	18.13	3.82	26.00
`computeAPSP`	2.11	1.39	7.00
`delete`	162.86	79.93	405.00
`insert`	137.36	43.48	300.00

5.6 Reproducibilidade de Experimentos

Todos os dados e códigos usados para os testes iniciais estão disponíveis em um repositório público. Os *datasets* incluem os modelos NML, topologias BRITE, códigos do *parsing*, consultas em Cypher do Neo4j, códigos da aplicação do MySQL e arquivos gnuplot.

5.7 Conclusões Parciais

Esse projeto de pesquisa apresenta 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). Com o trabalho desenvolvido até o momento, 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 iniciais, 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, por utilizar padrões da Web Semântica, o que 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 nas seguintes 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.

De acordo com o cronograma apresentado no Capítulo 4 as próximas atividades serão: (*i*) realizar experimentos finais e (*ii*) escrita da dissertação. Esta pesquisa resultou em trabalhos publicados em workshops ([22], [23]).

Capítulo 6

Metodologia

A metodologia para verificar a aplicabilidade da proposta se baseia na análise de primitivas existentes para contexto SDN, no estudo da linguagem de modelagem de topologia de redes compatível com Web Semântica e estudo de bancos de dados baseado em grafos e suas linguagens de consulta. A partir do conhecimento gerado, o objetivo é realizar testes que validem os objetivos apresentados no Capítulo 2.

6.1 Plano de Trabalho e Cronograma

Para o desenvolvimento dessa pesquisa, foram previstas as seguintes tarefas:

1. Pesquisa bibliográfica e definição dos itens de pesquisa;
2. Estudo do Banco de Dados Baseado em Grafos e Linguagens de Consulta;
3. Estudos do Modelo Semântico e Redes Definidas por Software;
4. Análise das limitações no modelo e no banco de dados baseado em grafo para o contexto de Redes Definidas por Software;
5. Desenvolvimento do parser para modelagem de topologias de rede;
6. Publicação e apresentação de artigo no WGRS15 com proposta inicial da arquitetura e resultados parciais.
7. Definição das primitivas a serem implementadas e experimentos a serem realizados;
8. Submissão e aceite de artigo no EWSDN15 com resultados do banco de dados relacional;
9. Exame de Qualificação de Mestrado;
10. Execução de novos experimentos e análise dos resultados;
11. Escrita da dissertação e defesa;

As tarefas foram organizadas por semestre, conforme cronograma apresentado na Tabela 6.1.

Tabela 6.1: Cronograma das tarefas

Objetivo	1^o **Semestre/2014**	2^o **Semestre/2014**	1^o **Semestre/2015**	2^o **Semestre/2015**
1	X			
2	X			
3	X	X		
4		X		
5			X	
6			X	
7			X	
8			X	
9			X	
10				X
11				X

6.2 Publicações

Com os estudos realizados até o momento, a pesquisa resultou em duas publicações:

1. Artigo "Modelos Semânticos em Bancos de Dados Baseados em Grafos para Aplicações de Controle de Redes Definidas por Software"apresentado e publicado nos anais do XX Workshop de Gerência e Operação de Redes e Serviços (WGRS) do XXXIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC) realizado em Vitória-ES em Maio de 2015[22].

2. Artigo "*Towards Semantic Network Models via Graph Databases for SDN Application*"aceito no *Fourth European Workshop on Software Defined Networks (EWSDN)* que será apresentado em Outubro de 2015 em Bilbao, Espanha [23].

Anexo A

Aplicação de Teste

A.1 Consultas Cypher

Conforme apresentado no Capítulo 4, as primitivas do NetGraph [17] foram escritas na linguagem de consulta *Cypher*, nativa do Neo4j. Abaixo serão apresentadas as consultas geradas e utilizadas pela aplicação descrita no Capítulo 5.

Grau de entrada de um *Node*:

```
1. MATCH (n:Node)<-[:hasInboundPort]-(p:Port)<-[isSink]-(l:Link)
2. WHERE n.name={nameNode}
3. RETURN COUNT(l) AS CountOutDegree
```

Grau de saída de um *Node*:

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

Vizinhos de um *Node*:

```
1. MATCH (n:Node)-[]-(p:Port)-[]-(l:Link)-[]-(p1:Port)-[]-(n2:Node)
2. WHERE n.name={nameNode}
3. RETURN DISTINCT(n2) AS Neighbors
```

Verificação da existência de rota entre dois *Nodes*:

```
1. MATCH p=shortestPath((n:Node)-[*]-(m:Node))
2. WHERE n.name = {nameNode} AND m.name={nameNode1}
3. RETURN COUNT(p) >0 AS DoesRouteExist
```

Cálculo de menor caminho de um *Node* para todos os outros:

```
1. MATCH (n:Node),(m:Node), p=shortestPath((n)-[*]->(m))
2. WHERE n.name={nameNode}
3. RETURN p
```

Cálculo de menor caminho de todos os pares de *Node*:

```
1. MATCH p=shortestPath((n:Node)-[*]->(m:Node))
2. RETURN p
```

Cálculo de k menores caminhos entre dois *Nodes*:

```
1. MATCH (n:Node),(m:Node), p=allShortestPaths((n)-[*]-(m))
2. WHERE n.name={nameNode} AND m.name={nameNode1}
3. RETURN p LIMIT {valueK}
```

Cálculo de *Minimum Spanning Tree* a partir de uma *Node* de origem:

```
1. MATCH p=shortestPath((n:Node)-[*]->(m:Node))
2. WHERE n.name = {nameNode}
3. RETURN p AS MinimumSpanningTree
```

Exclusão de um *Node* (e suas *Ports*):

```
1. MATCH (n:Node)-[r1]-(p:Port)-[r2]-(l:Link)-[r3]-(p2:Port)
2. WHERE n.name={nameNode}
3. DELETE n,r1,p,r2,l,r3
```

Atribuição de custo a um *Link*:

```
1. MATCH (n:Node)-[:hasOutboundPort]-()-[r]-(l:Link) 2. WHERE n.name={nameNode} AND l.na
3. SET l.cost={valueCost}
```

Busca de custo de um *Link*:

```
1. MATCH (n:Node)-[:hasOutboundPort]-()-[r]-(l:Link)
2. WHERE n.name={nameNode} AND l.name={nameLink}
3. RETURN l.cost
```

Inclusão de um *Node* (e suas *Ports*):

```
1. CREATE (n1:Node{name: newNode})
2. CREATE (n2:Port{name: portInNewNode})
3. CREATE (n3:Port{name: portOutNewNode})
4. WITH n1, n2, n3
5. CREATE (n1)<-[r:hasInboundPort]-(n2)
6. CREATE (n1)-[r2:hasOutboundPort]->(n3)
7. RETURN r,r2
```

Bibliografia

[1] T. K. et al., "Onix: A distributed control platform for large-scale production networks.," in *OSDI*, vol. 10, pp. 1–6, 2010.

[2] N. McKeown et al., "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

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

[4] NOVI, *Networking innovations Over Virtualized Infrastructures*. 2010.

[5] Escalona et al., "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*, pp. 1–8, 2011.

[6] P. Grosso, L. Herr, N. Ohta, P. Hearty, and C. de Laat, "Cinegrid: Super high definition media over optical networks," *Future Gener. Comput. Syst.*, vol. 27, pp. 881–885, July 2011.

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

[8] M. Aertsen, "Verifying functional requirements in multi-layer networks: a case for formal description of computer networks," March 2014.

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

[10] V. Soundararajan and S. Kakaraddi, "Applying graph databases to cloud management: An exploration," in *Cloud Engineering (IC2E)*, pp. 544–549, March 2014.

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

[12] NOSQL, "http://nosql-database.org/," 2015.

[13] I. Robinson, J. Webber, and E. Eifrem, *Graph Databases*. O'Reilly Media, Inc., 2013.

[14] S. Jouili and V. Vansteenberghe, "An empirical comparison of graph databases," in *Social Computing (SocialCom), 2013 International Conference on*, pp. 708–715, Sept 2013.

[15] F. Holzschuher and R. Peinl, "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, (New York, NY, USA), pp. 195–204, ACM, 2013.

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

[17] R. Raghavendra, J. Lobo, and K.-W. Lee, "Dynamic graph query primitives for sdn-based cloudnetwork management," in *Proceedings of Hot Topics in Software Defined Networks*, HotSDN '12, pp. 97–102, 2012.

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

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

[20] MySQL, "http://www.mysql.com/," 2015.

[21] Literateprograms, "Dijkstra's algorithm (java)," 2015.

[22] T. P. C. Souza, M. A. S. Santos, L. B. Paula, and C. E. Rothenberg, "Modelos semânticos em bancos de dados baseados em grafos para aplicações de controle de redes definidas por software," *XXXIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC2015): XX Workshop de Gerência e Operação de Redes e Serviço (WGRS*, 2015.

[23] T. P. C. Souza, M. A. S. Santos, L. B. Paula, and C. E. Rothenberg, "Towards semantic networks models via graph databases for sdn applications," *Europe Workshop on Software Defined Networks (EWDSN 2015)*, 2015.